UNIVERSIDAD NACIONA DE PIURA

ÁREA ACADEMICA DE LA FACULTAD DE CIENCIAS

ESCUELA PROFESIONAL DE INGENIERIA ELECTRONICA Y TELECOMUNICACIONES

PROYECTO DE INVESTIGACIÓN PARA OPTAR EL TÍTULO DE INGENIERO ELECTRONICO Y TELECOMUNICACIONES

"DISEÑO, IMPLEMENTACION Y PRUEBA DE UN ACELEROMETRO PARA OBTENER DESPLAZAMIENTO"

PERSONAL INVESTIGADOR:

BACH. CARMEN AGUIRRE ROOSEBELT YONATAN

ASESOR:

PhD. ANTENOR ALIAGA ZEGARRA

PIURA – PERÚ

2016

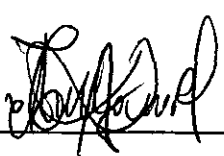
Los miembros del jurado designados para evaluar la tesis presentada por el Bachiller Carmen Aguirre Roosebelt Yonatan, titulada:

"DISEÑO, IMPLEMENTACION Y PRUEBA DE UN ACELEROMETRO PARA OBTENER DESPLAZAMIENTO"

Considera que la misma cumple con los requisitos exigidos para alcanzar al Título de Ingeniero Electrónico y Telecomunicaciones.

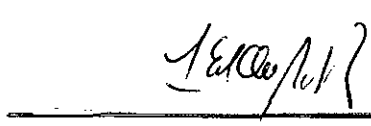
Ing. Jacinto Sandoval Juan Manuel

Presidente



Ing. Arellano Ramírez Carlos Enrique

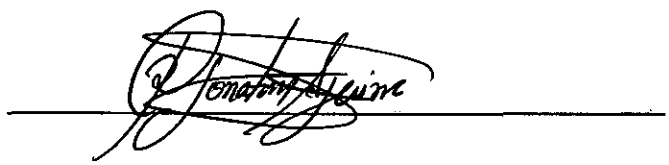
Secretario



Ing. Ávila Regalado Eduardo Omar

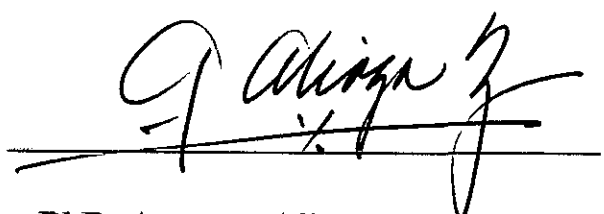
Vocal

"DISEÑO, IMPLEMENTACION Y PRUEBA DE UN ACELEROMETRO PARA OBTENER DESPLAZAMIENTO"



Br. Carmen Aguirre Roosebelt Yonatan

Ejecutor de Tesis



PhD. Antenor Aliaga Zegarra

Asesor

INTRODUCCION

A medida que el tiempo pasa, el avance de la tecnología ha permitido al hombre diseñar sistemas que cada día facilitan más nuestro estilo de vida con mejoras importantes, las cuales se han dado en diferentes áreas y de hecho en una de las importantes como la electrónica.

Con el desarrollo tecnológico; tanto a nivel de hardware como software, los dispositivos electrónicos son cada vez más sofisticados y tienden a la miniaturización, sumado a que a los sensores cada vez más se les dota de inteligencia. Actualmente, es cada vez más necesaria la implementación de nuevas tecnologías, sobre todo en el campo Industrial, como lo es la industria petrolera en donde la exigencia del uso de sistemas confiables para la adquisición de datos resulta esencial para la construcción de sistemas de monitoreo permanente de los equipos de bombeo mecánico. El diagnóstico de los equipos de bombeo se realiza mediante pruebas dinamométricas registradas en la superficie, las cuales nos brindan información de la operación del pozo. Un dinamómetro suministra datos de desplazamiento y fuerzas de la barra pulida, en la superficie del pozo. Utilizando estos datos se puede obtener las fuerzas y el desplazamiento de la bomba en el fondo del pozo, para determinar la operación de la bomba, calcular la producción del pozo y prevenir fallas de operación.

En este proyecto de investigacion desarrollamos una aplicación para medir el desplazamiento de la varilla de succión; la cual es una variable importante en el diagnóstico, mediante la utilización de un acelerómetro y un microcontrolador. Además acorde a la vanguardia tecnológica se incorpora la utilización de la comunicación Bluetooth de la mano con la comunicación USB, para realizar un monitoreo en tiempo real del desplazamiento de la varilla pulida.

RESUMEN

Este proyecto de investigación desarrolla un sistema que permite medir el desplazamiento de la varilla pulida de succión en un equipo de bombeo mecánico utilizado en los pozos petroleros. Este desplazamiento se mide a partir de datos de aceleración que provienen de un acelerómetro que se encuentra en una tarjeta de adquisición montada en la cabeza del equipo de bombeo. Esta tarjeta llamada Slave, consta de un microcontrolador que se comunica con el acelerómetro y envía los datos de aceleracion mediante comunicación inalámbrica Bluetooth a otra tarjeta llamada Gateway que cumple el papel de ser el enlace entre el Slave y el Master que será nuestra PC, donde se encuentra nuestra aplicación desarrollada en la interfaz de diseño gráfico (GUIDE) del Software Matlab, para llevar a cabo la comunicación y presentación del desplazamiento.

El Gateway consta de un microcontrolador que posee el periférico USB, con el cual establece comunicación con el Master, y lógicamente un Bluetooth para poder recibir y enviar los paquetes de comunicación desde y hacia el Slave. Para tener un nivel de confiabilidad en los datos, se diseñó un protocolo de comunicación que asegure la integridad de los datos, la secuencia y que este en las condiciones de hacerle frente a los problemas que puedan presentarse en el proceso de comunicacion. Con un nivel de confiabilidad en los datos se realiza el procesamiento de la data que consiste en realizar una doble integral de la aceleración, lógicamente complementados de técnicas de filtrado digital y métodos de integración numérica para obtener finalmente el desplazamiento.

ABSTRACT

This research project is to develop a system for measuring the displacement of the rod suction equipment mechanical pumping from acceleration data coming from an accelerometer located on an acquisition card mounted on the head of the team pumping, which describes a movement up and down. This card called Slave, consists of a microcontroller that communicates with the accelerometer and sends the data via Bluetooth wireless communication to another card called Gateway that fulfills the role of being the link between the Slave and the Master will be our PC, where it is our application developed in the graphical design interface (GUIDE) Software Matlab, to perform communication and display offset.

The Gateway consists of a microcontroller that has the USB peripheral with which establishes communication with the Master, and obviously a Bluetooth to receive and send communication packets to and from the Slave. To have a level of reliability in the data, a communication protocol that ensures the integrity of desing data, sequence and that the conditions to cope with problems that may arise in the channel was designed communications. With a level of reliability in data processing data that is to perform a double integral of acceleration, logically complemented digital filtering techniques and numerical integration methods to finally get the movement is made.

DEDICATORIA

Primeramente el agradecimiento a dios, porque sin su infinito amor y bondad que nos brinda cada dia de nuestras vidas, este trabajo no hubiese sido posible, ni tampoco haber escalado un peldaño mas en nuestra vida.

A mis padres Rolando, Carla, Maria y Alberto, que supieron ser mi respaldo, guía y motivo de aliento cada dia, por inculcarme el deseo de superación y el anhelo de triunfar, que me permitio emprender este sueño que ahora puedo ver concluido, todo esto es por ustedes y para ustedes. A mi mama Carla que es mi motor y motivo para seguir adelante cada dia, gracias por tantas desveladas, por tantas atenciones conmigo y por sacar fuerzas de empuje para sacarme adelante. A mi papa Alberto, quien me apoya cada dia a salir adelante. Gracias también a una persona en especial, que hoy ya no esta físicamente conmigo, pero que siempre esta en mi corazón, mi papá Rolando. Se que mis palabras también no llegaran al cielo en donde estas pero quiero darte gracias por regalarme 20 años de tu vida a tu lado, gracias por todo tu amor, aliento, apoyo incondicional y por acompañarme en este sueño que comenzamos juntos; por motivos del destino me dejaste en el camino pero se que si estuvieses conmigo sentirías la misma alegría que siento yo en este momento de mi vida porque este logro es en gran parte por ti.

A mis hermanos Rubi, Cinthia, Ericka, Yessica, Mariela, Miguel, Franklin, Luis y Sugey, a mis primos Betty, Robert, Marita y Flor y tios Rosa, Maria, Leoncio, Chabuca; y a toda mi familia porque siempre me alentaron en esta carrera de largo aliento, y de alguna manera aportaron con sus consejos y palabras de motivación, para salir adelante.

Las palabras no alcazarian para agradecerles por todo su apoyo y comprensión en momento difíciles. Muchas gracias de corazón.

AGRADECIMIMENTOS

Este proyecto no se podría haber llevado a cabo principalmente sin el apoyo y colaboración de muchas personas, a las cuales les expreso mi más sincera gratitud y agradecimiento de corazón.

Primero y como mas importante, un agradecimiento especial a mi asesor y mentor, Phd. Antenor Segundo Aliaga Zegarra. Gracias por brindarme sus conocimientos, orientación y dedicacion no solo en el ámbito académico; en el cual ha contribuido a mi formación profesional inculcándome un espíritu de investigación constante, sino también en el plano personal. Mi eterno agradecimiento por brindarme su amistad, por todos los consejos y por el apoyo desinteresado que me ha brindado cada dia a lo largo de estos casi cuatro años, del cual estare siempre en deuda. A usted mi lealtad y admiración.

Una mención especial a todos los ingenieros que fueron parte de mi formación como alumno y como egresado, asi también a mis profesores Lidia Rojas y Gabriel Estrada los cuales fueron pieza clave para que el sueño de ingresar a la universidad fuese posible. Gracias por sus conocimientos, apoyo y predisposición mostrada hacia mi persona en todo momento.

Mi agradecimiento también y no menos importante a las personas que sin llevar tu sangre son parte de tu familia de corazon y esos son mis amigos, que de hecho también son parte fundamental de este logro. Un agradecimiento especial para mis fieles amigos "indicados", ING. Hebert Espino Aguirre y Moises Calle Puelles, gracias por ser parte del dia a dia, por las experiencias compartidas y por su apoyo incondicional en este logro. Agradecer también a mis amigos David Saldarriaga Castillo y Luis Seminario por su apoyo desde nuestra etapa de estudiantes hasta hoy. A mi amigo y hermano del alma Luis Alvarado Saldarriaga, gracias por todas las ocasiones en que me has ayudado a conseguir mis logros y por brindarme tu amistad sincera a lo largo de los años. Tambien un agradecimiento a mis amigas infaltables Diana Saba Pozo, Lissete Zapata Loayza, Yanet Morales Guerrero; a quienes conoci en mi etapa de estudiante y me brindaron su amistad sincera y estuvieron conmigo en los buenos y malos momentos, a mi hermana de corazón Raquel Palomino Romero, que siempre me a contagiado con esas ganas de salir adelante, y finalmente a mi socia, colega y amiga Roxana Ordinola; la cual siempre me motiva y celebra mis logros como yo los de ella, y a todas la personas que contribuyeron en algo, muchas gracias por sus muestras de aliento y apoyo constante para alcanzar esta meta.

CONTENIDO

CAPITULO I

PLANTEAMIENTO METODOLOGICO

1.1. Descripción de la realidad problemática

Actualmente el bombeo mecánico es uno de los métodos de producción más utilizados entre un 80 y 90 % para la extracción del crudo. Su principal característica es la de utilizar una unidad de bombeo para transmitir movimiento al pistón de la bomba de subsuelo a cierta profundidad del fondo del pozo a través de una sarta de varillas de succión y mediante la energía suministrada por un motor.

Para poder realizar un monitoreo, así como estimar algún déficit en la producción, existe la necesidad de medir el desplazamiento de la bomba sumergible, como la fuerzas de tensión y compresión en la varilla de succión para obtener registros de cargas y desplazamiento contra el tiempo que permitan obtener una carta dinamométrica que pueda ser interpretada y que nos describa el comportamiento del equipo de bombeo.

Actualmente existen dinamómetros, los cuales se colocan en la varilla pulida, como los de la marca Delta II los cuales tienen integrados un transductor de cargas de alta sensibilidad así como un transductor de desplazamiento el cual está basado en un servomecanismo que acciona la bobina de un potenciómetro, los cambios tanto de potencial y de resistencia son guardados en un registrador de dos canales en forma de desplazamiento vs tiempo. Lógicamente que el método para medir el desplazamiento estará afectado por el desgaste del cursor y la resistencia producidos por el rozamiento, la variación de la resistencia o la abrasión. Existen otros que incluyen tecnología GPRS pero su costo no es muy accesible bordeando aproximadamente los $2500, por lo que el presente proyecto propone diseñar una nueva solución confiable, práctica y accesible para la medida del desplazamiento del movimiento periódico que describe la varilla de succión, mediante la utilización de un acelerómetro digital; los cuales están presentes en múltiples aplicaciones en la actualidad; y aprovechando que actualmente las comunicaciones inalámbricas, sensores, procesadores y software para procesamiento de data están a nuestro alcance desarrollar este sistema de medición.

1.1. Delimitación y formulación del problema

El problema principal radica en que actualmente, es cada vez más necesaria la implementación de nuevas tecnologías, sobre todo en el campo Industrial y el presente proyecto de investigación apunta a desarrollar un prototipo ó piloto que permita en este caso monitorear el desplazamiento de la varilla de succión del equipo de bombeo mecánico de petróleo a partir de datos de aceleración utilizando un acelerómetro digital y un microcontrolador, el cual se comunicara a una placa central mediante comunicación Bluetooth. Esta placa central de adquisición nos permitirá comunicarnos mediante protocolo de comunicación USB a nuestro PC o Laptop para finalmente ayudados del software Matlab realizar el procesamiento, cálculos numéricos y presentación del desplazamiento en un GUIDE (editor de interfaces de usuario-GUI).

En cuanto a Delimitación, el presente proyecto está orientado a ser implementado para en cualquier tipo de equipo de bombeo mecánico que pueda ser sometido a parámetros de Control y de aplicaciones Industriales, pero para efectos de demostración y sustentación a la finalización de la Investigación, estará delimitado en una maqueta a escala que permita simular el movimiento que describe la varilla de succión de arriba hacia abajo (movimiento armonico simple) en un pozo real.

1.2. Formulación del problema

¿Es posible diseñar un sistema que nos permita medir el desplazamiento del movimiento periódico que describe un objeto a partir de datos de aceleración?

1.3. Objetivos

1.3.1. Objetivo general

- Diseñar e implementar un sistema que permita medir el desplazamiento de un movimiento periodico a partir de datos de aceleración que provenien de un acelerómetro digital.

1.4.2. Objetivos específicos

- Seleccionar un acelerómetro como sensor.
- Seleccionar el microcontrolador para obtener la aceleración.
- Diseñar hardware y software para obtener la data.
- Implementar la comunicación inalámbrica usando la tecnología Bluetooth 4.0.

- Utilizar la comunicación USB y enviar la data obtenida a la PC.
- Procesar la data, en este caso aceleración, para obtener el desplazamiento y presentarla en la PC.

1.5. Hipótesis de la investigación

La utilización de un acelerómetro como sensor en combinación con las tecnologías existentes, permitirá el diseño de un sistema para obtener la medida del desplazamiento de un movimiento periódico a partir de datos de aceleracion.

1.6. Justificación de la investigación

Existen diferentes tipos de sensores que son utilizados en la industria de control para medir desplazamiento como lo son los LVDT (Linear Variable Differencial Transformer), sensores de medición laser, dependiendo de su aplicación, o potenciómetros, pero la elección de utilizar un acelerómetro con el propósito de medir el desplazamiento de un movimiento no es muy común.

Una de las razones importantes que motiva el desarrollo de nuestro trabajo de investigación es que existe la necesidad de realizar un monitoreo de producción en la extracción de crudo en los pozos petroleros para lo cual es necesario medir además las fuerzas de tensión y compresión de la varilla de succión o también llamada barra pulida; medir el desplazamiento de la bomba sumergida por lo tanto se mide el desplazamiento de la varilla pulida que se encuentra en la superficie, la cual describe un movimiento periodico para obtener curvas desplazamiento vs tiempo, y así poder desarrollar parte del sistema que permita el monitoreo para mejorar la operación de equipos de bombeo mecánico y prever futuros problemas de producción.

Es por esto que surge la idea de realizar el presente trabajo de investigación centrándonos en la medición del desplazamiento basado en el estudio del comportamiento de un acelerómetro para utilizarlo como sensor y en combinación con las diferentes herramientas que nos ofrece la tecnología electrónica (hardware-software) diseñar e implementar un sistema que permita medir desplazamiento a partir de datos de aceleración.

1.7. Importancia de la investigación

El diseño y desarrollo de una nueva forma de medir desplazamiento utilizando las tecnologías existentes genera gran interés en el investigador, debido a la factibilidad para su desarrollo, por ser un tema de actualidad tecnológica que permite el desarrollo de las capacidades y destrezas investigativas generando una proyección al futuro en el campo industrial siendo uno de los principales beneficiados en primer lugar la industria petrolera, ya que se optimizarán recursos con un producto que les asegure confiabilidad y a un costo no demasiado elevado, que les permite un mejor desempeño en la supervisión del funcionamiento del equipo de bombeo y prever problemas que mermen la producción, con los cual lograrían evitar pérdidas y a su vez satisfacer a los clientes cubriendo las necesidades que en la actualidad demandan los propietarios de las empresas.

1.8. Limitaciones de la investigación

Las posibles limitaciones que se pueden dar:

- Una de las limitaciones es la poca información de Sistemas de medición de desplazamiento utilizando acelerómetros a pesar de que estos están presentes en la mayoría de aplicaciones e componentes de nuestra vida cotidiana.
- Para efectos de Implementación real, una limitación es la poca confianza de las empresas en aplicaciones que no están respaldadas por empresas de renombre internacional.

1.9. Variables

1.9.1. Variables dependientes

- Desplazamiento

1.9.2. Variable independiente

- Aceleración

1.10. Viabilidad de la investigación

1.10.1. Viabilidad técnica

El proyecto se considera factible técnicamente ya que la adquisición de equipos tecnológicos que se necesitan para desarrollar el proyecto de investigación está disponibles en el mercado y al alcance del investigador, por lo que la barrera tecnológica no es un problema para desarrollar nuestro trabajo de investigación.

1.10.2. Viabilidad operativa

Para poder llevar a cabo el proyecto de investigación se requiere tener una idea de cómo opera un pozo en tiempo real, el número de strokes, velocidad de bombeo entre otros aspectos, y para esto se cuenta con las facilidades para el acceso para la observación en campo de la operatividad del proceso en estudio. Por lo tanto nuestro proyecto se considera factible operativamente nuestro proyecto ya que se puede observar el desempeño en tiempo real del equipo de bombeo.

1.10.3. Viabilidad económica

El proyecto es factible económicamente ya que el presupuesto formulado en el trabajo de investigación no es de un costo elevado y permite que el investigador desarrolle el proyecto con recursos propios.

CAPITULO II:

MARCO TEORICO CONCEPTUAL

2.1. Antecedentes de la investigación

Trabajo de Investigación - Universidad Tecnológica de Pereira. (2004)

"***Determinación de la aceleración, velocidad y desplazamiento utilizando acelerómetros micro maquinados***". Este trabajo de investigación presenta una metodología para medir cargas inerciales basadas en las oscilaciones de una plataforma en la cual se suspendida una barra elástica. Se realizó la simulación de la medición de la aceleración, velocidad y desplazamiento mediante la adquisición de datos utilizando una tarjeta análoga modelo NI6023E y el programa LABVIEW.

Trabajo final de carrera - Centro Politécnico de la Universidad de Zaragoza. (2007)

"***Análisis de acelerómetros comerciales como sensores para medir desplazamientos***". En el presente trabajo se realizó el análisis de tres acelerómetros diferentes para medir el desplazamiento de un carrito en movimiento a partir de su aceleración. Se utilizó como circuito de control la plataforma de desarrollo de hardware libre Arduino y para establecer la comunicación inalámbrica entre el Arduino montado en el carrito y la PC se optó por el uso de módulos ZigBee. Finalmente el diseño de software para presentar el desplazamiento fue realizado en Processing.

Tesis para obtener el grado de Ingeniero Electrónico - Universidad Austral de Chile (2008)

"***Diseño de un acelerómetro utilizando tecnología de navegación óptica a partir de un mouse para computador***". En el presente trabajo se elaboró un acelerómetro que detecta el desplazamiento de una masa en el extremo de una viga en voladizo utilizando una fotografía del reflejo de una luz led sobre una superficie reflejada en un sensor CMOS el cual envía las imágenes a un DSP y mediante software procesa matemáticamente los datos y obtiene la velocidad y desplazamiento instantáneo de la masa.

2.2. Marco conceptual

2.2.1. Sensores

Un sensor es un dispositivo que permite transformar señales físicas o químicas, llamadas variables de instrumentación, en señales eléctricas las cuales podemos manipular.

La señal eléctrica puede ser resistencia eléctrica (con una RTD), capacidad eléctrica (con un sensor de humedad), tensión eléctrica (un termopar), corriente eléctrica (fototransistor), etc.

También puede decirse que son dispositivos que convierten una forma de energía en otra y pueden estar conectados a un computador para obtener ventajas como son el acceso a una base de datos, la adquisición de datos de un proceso, etc.

2.2.2. Componentes de un sensor

2.2.2.1. Captador

Dispositivo con un parámetro **"p"** que es sensible a una magnitud física **"h"** – emite energía **"w"** que depende de **"p"** (y de **"h"**).
Idealmente seria: **w(t) = K*h(t),** donde K es una constante.

2.2.2.2. Transductor

Recibe la energía **w** del captador y la transforma en energía eléctrica **e(t)** y la retransmite.

2.2.2.3. Acondicionador

Recibe la señal de **e(t)** del transductor y la ajusta a niveles de voltaje o intensidad, precisos para su posterior tratamiento, dando **V(t).**

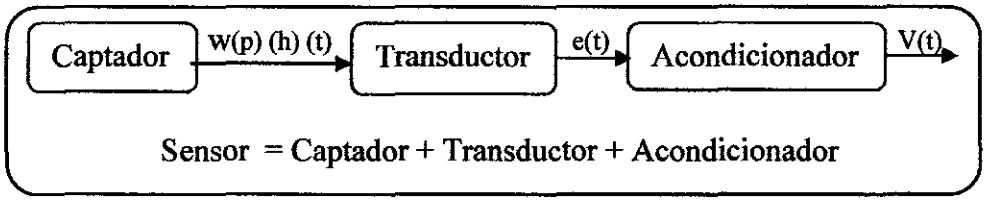


Figura 2. 1 Diagrama de bloques de un sensor

2.2.3. Características técnicas de un sensor

- **Rango de medida:** Conjunto de valores máximos y mínimos para las variables de entrada y salida, representados por una magnitud de medida en el que puede aplicarse el sensor.
- **Exactitud:** Se refiere a la desviación de la lectura de un sistema de medida respecto a una entrada conocida. El mayor error esperado entre las señales medida e ideal
- **Offset o desviación de cero:** Valor de la variable de salida cuando la variable de entrada es nula.
- **Linealidad:** es también llamada correlación lineal y quiere decir que si se tienen dos variables (A y B) existirá correlación si al aumentar los valores de A lo hacen también los de B y viceversa.
- **Sensibilidad:** Es la relación entre la variación de la señal de salida y la variación de la señal de entrada.
- **Resolución:** La mínima variación de la señal de entrada que puede apreciarse a la salida.
- **Rapidez de respuesta:** tiempo que depende de cuánto varíe la señal a medir. También depende de la capacidad del sistema para seguir las variaciones de la señal de entrada.
- **Excitación:** Es la cantidad de corriente o voltaje requerida para el funcionamiento del sensor.

2.2.4. Clasificación de sensores

Los sensores pueden ser clasificados por diferentes criterios:

2.2.4.1. Según el principio de funcionamiento

1) **Pasivos**

   Son aquellos que generan señales representativas de las magnitudes a medir por intermedio de una fuente auxiliar.

2) **Activos**

   Son aquellos que generan señales representativas de las magnitudes a medir de forma autónoma, sin requerir fuentes de alimentación.

2.2.4.2. Según el tipo de señal de salida

1) **Sensores Analógicos**

Un sensor analógico es aquel que puede entregar una salida continua variable dentro de un determinado rango, por ejemplo voltaje o corriente eléctrica, como se muestra en la figura.

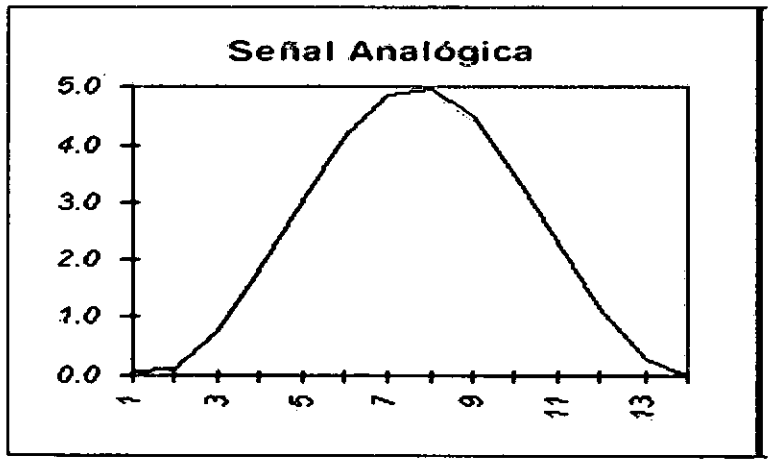


Figura 2. 2 Señal de salida entregada por un sensor analógico

2) **Sensores Digitales**

Un sensor digital es aquel que entrega una salida del tipo discreta, es decir que el sensor posee una salida que varía dentro de un determinado rango de valores, pero a diferencia de los sensores analógicos, esta señal varia de a pequeños pasos pre-establecidos.

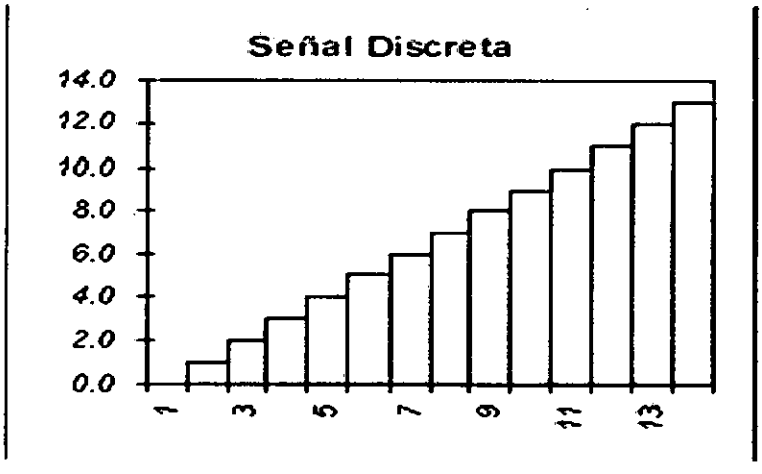


Figura 2. 3 Señal de salida entregada por un sensor digital

2.2.4.3. Según el tipo de magnitud a medir

Esta clasificación subdivide a los sensores de acuerdo a la magnitud física por medir tal es el caso de caudal, presión, nivel, entre otros.

1) **Temperatura**
   - Termistor
   - Termopar
   - Pirómetro
2) **Presión**
   - Barómetros
   - Manómetros
3) **Fuerza**
   - Galgas extensiométricas
   - Dinamómetro
   - Piezoeléctrico
4) **Humedad**
5) **Nivel**
   - Ultrasonido
   - Capacitivos
6) **Aceleración**
   - Acelerómetros
7) **Presencia o proximidad**
   - Inductivos
   - Capacitivos
   - Hall
   - Infrarrojos
   - Laser
8) **Caudal**
   - Medidores magnéticos
   - Medidores por fuerzas de coriolisis
   - Medidores de área variable
9) **Desplazamiento**
   - LVDT
   - Potenciómetros lineales

2.2.5. Acelerómetros

Los acelerómetros son dispositivos electromecánicos utilizados para medir las fuerzas de aceleración. Estas fuerzas pueden ser estáticas, como la constante de gravedad que nos empuja al centro de la Tierra, o dinámicas, como el movimiento o la vibración del acelerómetro. Estos dispositivos convierten la aceleración de la gravedad o de movimiento en una señal eléctrica analógica proporcional a la fuerza aplicada al sistema, o mecanismo sometido a vibración o aceleración. Esta señal que nos entrega el sensor (analógica o digital), indica en tiempo real la aceleración instantánea del objeto sobre el cual el acelerómetro está montado. (Manzanares del Moral, 2007).

Las técnicas convencionales para detectar y medir la aceleración se fundamenta en las leyes del movimiento de Newton, específicamente en la segunda ley conocida como Ley Fundamental de la Dinámica. Esta ley dada por la ecuación 3.1, la cual se encarga de cuantificar el concepto de fuerza. La aceleración que adquiere un cuerpo es proporcional a la fuerza neta aplicada sobre el mismo. La constante de proporcionalidad es la masa del cuerpo (que puede ser o no ser constante). Entender la fuerza como la causa del cambio de movimiento y la proporcionalidad entre la fuerza impresa y el cambio de la velocidad de un cuerpo es la esencia de esta segunda ley.

$$\mathbf{F = m * a} \qquad \ldots \text{ Ecuación 3.1}$$

Donde:

F: fuerza aplicada

a: aceleración

m: masa.

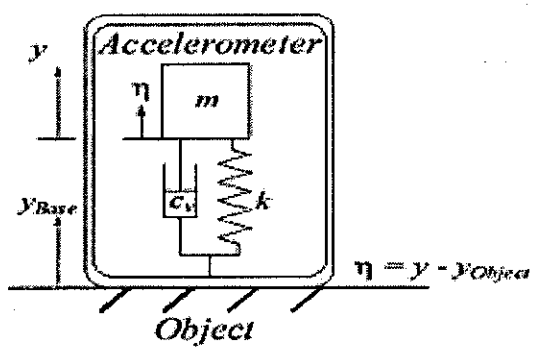


Figura 2. 4 Acelerómetro mecánico

2.2.5.1. Tipos de acelerómetros

Existen diferentes tipos de acelerómetros que varían de acuerdo al tipo de material del cual se fabrican estos dispositivos, entre los cuales tenemos:

2.2.5.1.1. Acelerómetros Mecánicos

Emplean una masa inerte y resortes elásticos. En este tipo de acelerómetro los cambios se miden con galgas extensiométricas, incluyendo sistemas de amortiguación que evitan la propia oscilación. Otros sistemas emplean sistemas rotativos desequilibrados que originan movimientos oscilatorios cuando están sometidos a aceleración (servo acelerómetros) o detectan el desplazamiento de una masa inerte mediante cambios en la transferencia de calor (acelerómetros térmicos). (Manzanares del Moral, 2007)

2.2.5.1.2. Acelerómetros Capacitivos

Estos dispositivos modifican la posición relativa de las placas de un micro condensador cuando está sometido a aceleración. El movimiento paralelo de una de las placas del condensador hace variar su capacidad.

Los acelerómetros capacitivos basan su funcionamiento en la variación de la capacidad entre dos ó más conductores entre los que se encuentra un dieléctrico, en respuesta a la variación de la aceleración.

Los sensores capacitivos en forma de circuito integrado en un chip de silicio se emplean para la medida de la aceleración. Su integración en silicio permite reducir los problemas derivados de la temperatura, humedad, capacidades parásitas, terminales, alta impedancia de entrada, etc.

Cuando se observa el sensor micro mecanizado parece una "H". Los delgados y largos brazos de la "H" están fijos al

2.2.5. Acelerómetros

Los acelerómetros son dispositivos electromecánicos utilizados para medir las fuerzas de aceleración. Estas fuerzas pueden ser estáticas, como la constante de gravedad que nos empuja al centro de la Tierra, o dinámicas, como el movimiento o la vibración del acelerómetro. Estos dispositivos convierten la aceleración de la gravedad o de movimiento en una señal eléctrica analógica proporcional a la fuerza aplicada al sistema, o mecanismo sometido a vibración o aceleración. Esta señal que nos entrega el sensor (analógica o digital), indica en tiempo real la aceleración instantánea del objeto sobre el cual el acelerómetro está montado. (Manzanares del Moral, 2007).

Las técnicas convencionales para detectar y medir la aceleración se fundamenta en las leyes del movimiento de Newton, específicamente en la segunda ley conocida como Ley Fundamental de la Dinámica. Esta ley dada por la ecuación 3.1, la cual se encarga de cuantificar el concepto de fuerza. La aceleración que adquiere un cuerpo es proporcional a la fuerza neta aplicada sobre el mismo. La constante de proporcionalidad es la masa del cuerpo (que puede ser o no ser constante). Entender la fuerza como la causa del cambio de movimiento y la proporcionalidad entre la fuerza impresa y el cambio de la velocidad de un cuerpo es la esencia de esta segunda ley.

$$\mathbf{F = m * a} \quad \ldots \text{ Ecuación 3.1}$$

Donde:

F: fuerza aplicada

a: aceleración

m: masa.

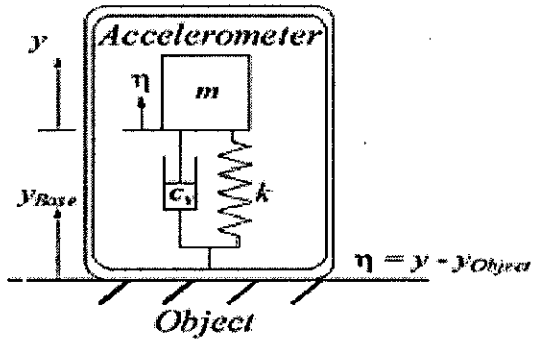


Figura 2. 4 Acelerómetro mecánico

2.2.5.1. Tipos de acelerómetros

Existen diferentes tipos de acelerómetros que varían de acuerdo al tipo de material del cual se fabrican estos dispositivos, entre los cuales tenemos:

2.2.5.1.1. Acelerómetros Mecánicos

Emplean una masa inerte y resortes elásticos. En este tipo de acelerómetro los cambios se miden con galgas extensiométricas, incluyendo sistemas de amortiguación que evitan la propia oscilación. Otros sistemas emplean sistemas rotativos desequilibrados que originan movimientos oscilatorios cuando están sometidos a aceleración (servo acelerómetros) o detectan el desplazamiento de una masa inerte mediante cambios en la transferencia de calor (acelerómetros térmicos). (Manzanares del Moral, 2007)

2.2.5.1.2. Acelerómetros Capacitivos

Estos dispositivos modifican la posición relativa de las placas de un micro condensador cuando está sometido a aceleración. El movimiento paralelo de una de las placas del condensador hace variar su capacidad.

Los acelerómetros capacitivos basan su funcionamiento en la variación de la capacidad entre dos ó más conductores entre los que se encuentra un dieléctrico, en respuesta a la variación de la aceleración.

Los sensores capacitivos en forma de circuito integrado en un chip de silicio se emplean para la medida de la aceleración. Su integración en silicio permite reducir los problemas derivados de la temperatura, humedad, capacidades parásitas, terminales, alta impedancia de entrada, etc.

Cuando se observa el sensor micro mecanizado parece una "H". Los delgados y largos brazos de la "H" están fijos al

substrato. Los otros elementos están libres para moverse, lo forman una serie de filamentos finos, con una masa central, cada uno actúa como una placa de un condensador variable, de placas paralelo.

La aceleración o desaceleración en el eje "SENSOR", ejerce una fuerza a la masa central. Al moverse libremente, la masa desplaza las minúsculas placas del condensador, provocando un cambio de capacidad. Este cambio de capacidad es detectado y procesado para obtener un voltaje de salida fácil de utilizar utilizando una tecnología BiCMOS (BiMOS II).

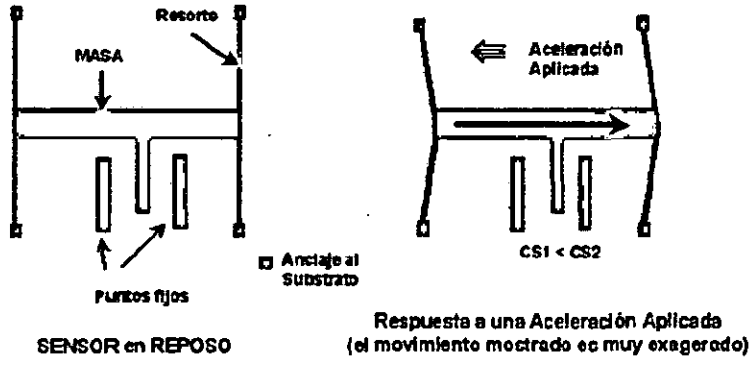


Figura 2. 5 Sistema de placas para un acelerómetro capacitivo

2.2.5.1.3. Acelerómetros Piezoeléctricos

Su funcionamiento se basa en el efecto piezoeléctrico. Una deformación física del material causa un cambio en la estructura cristalina y así cambian las características eléctricas. Su principal inconveniente radica en su frecuencia máxima de trabajo y en la incapacidad de mantener un nivel permanente de salida ante una entrada común. (Gonzalo Arribas, 2005)

La mecánica consiste en pegar un cristal piezoeléctrico a una masa conocida para que cuando se encuentre sometido a vibración se genere una fuerza que actúa sobre el acelerómetro.

Esta fuerza es el resultado de la aceleración por la masa sísmica. Debido a que la masa sísmica es constante se obtiene una señal proporcional a la aceleración de la misma.

La presencia de un pre-amplificador es cada vez más habitual ya que producen un valor de tensión proporcional a la excitación aplicada en la salida del amplificador y su comportamiento resulta independiente del conexionado exterior puesto que carga y resistencia de entrada del amplificador se mantiene constante siempre, pero para esto el sensor precisa alimentación.

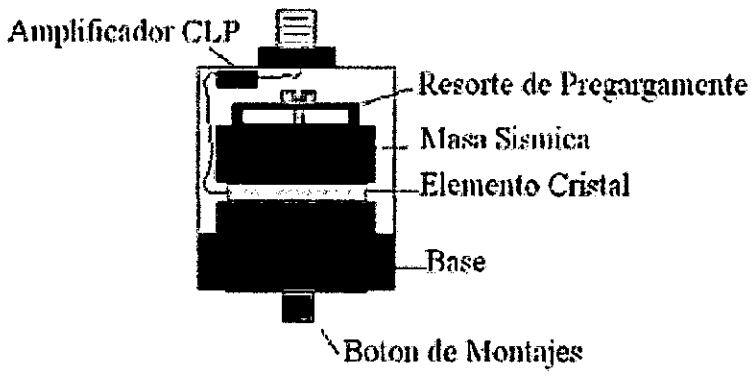


Figura 2. 6 Acelerómetro Piezoeléctrico

2.2.5.1.4. Acelerómetros micro mecánicos

Tambien llamados **MEMS** según las siglas anglosajonas *Micro-Electro-Mechanical-System.* Este tipo de dispositivos ha sido desarrollado para su empleo como sensor de impacto en los sistemas de airbag, en sistemas antibloqueo de frenos o en cualquier otro proceso en que se pretenda medir impacto. (Gonzalo Arribas, 2005)

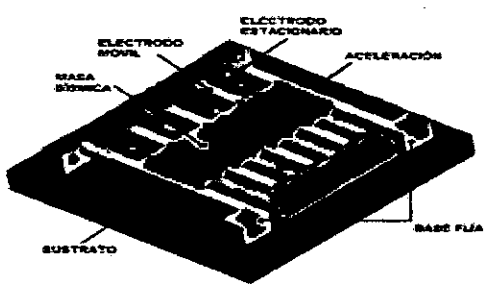


Figura 2. 7 Micro acelerómetro

CAPITULO III

PROTOCOLOS DE COMUNICACION

3.1 Definición de protocolo

En informática y telecomunicaciones, un protocolo de comunicación es un sistema de reglas que permiten que dos o más entidades de un sistema de comunicación se comuniquen entre ellas para transmitir información por medio de cualquier tipo de variación de una magnitud física. Se trata de las reglas o el estándar que define la sintaxis, semántica y sincronización de la comunicación, así como también los posibles métodos de recuperación de errores. Los protocolos pueden ser implementados por hardware, por software, o por una combinación de ambos.

Los sistemas de comunicación utilizan formatos bien definidos (protocolo) para intercambiar mensajes. Cada mensaje tiene un significado exacto destinado a obtener una respuesta de un rango de posibles respuestas predeterminadas para esa situación en particular. Normalmente, el comportamiento especificado es independiente de cómo se va a implementar. Los protocolos de comunicación tienen que estar acordados por las partes involucradas. Para llegar a dicho acuerdo, un protocolo puede ser desarrollado dentro de estándar técnico. Los protocolos de comunicación permiten el flujo información entre equipos que manejan lenguajes distintos, por ejemplo, dos computadores conectados en la misma red pero con protocolos diferentes no podrían comunicarse jamás, para ello, es necesario que ambas ***"hablen"*** el mismo idioma. Un lenguaje de programación describe el mismo para los cálculos, por lo que existe una estrecha analogía entre los protocolos y los lenguajes de programación: «los protocolos son a las comunicaciones como los lenguajes de programación son a los cómputo. Un protocolo de comunicación, también llamado en este caso protocolo de red, define la forma en la que los distintos mensajes o tramas de bit circulan en una red de computadoras.

Por ejemplo, el protocolo sobre palomas mensajeras permite definir la forma en la que una paloma mensajera transmite información de una ubicación a otra, definiendo todos los aspectos que intervienen en la comunicación: tipo de paloma, cifrado del mensaje, tiempo de espera antes de dar a la paloma por 'perdida'... y cualquier regla que ordene y mejore la comunicación.

Si bien los protocolos pueden variar mucho en propósito y sofisticación, la mayoría especifican una o más de las siguientes propiedades:

- Detección de la conexión física subyacente (con cable o inalámbrica), o la existencia de otro punto final o nodo.
- *Handshaking*: proceso automatizado de negociación que tiene lugar cuando un equipo está a punto de comunicarse con un dispositivo exterior para establecer las normas para la comunicación.
- Negociación de varias características de la conexión.
- Cómo iniciar y finalizar un mensaje.
- Procedimientos en el formateo de un mensaje.
- Qué hacer con mensajes corruptos o formateados incorrectamente (corrección de errores).
- Cómo detectar una pérdida inesperada de la conexión, y qué hacer entonces.
- Terminación de la sesión y/o conexión.
- Estrategias para mejorar la seguridad (autenticación, cifrado).

3.2 Protocolo de comunicación RS232

El puerto serial de las computadoras es conocido como puerto RS-232, la ventaja de este puerto es que todas las computadoras traen al menos un puerto serial, este permite la comunicaciones entre otros dispositivos tales como otra computadora, el mouse, la impresora y para nuestro caso con los microcontroladores. Existen dos formas de intercambiar información binaria: la paralela y el serial.

La comunicación paralela transmite todos los bits de un dato de manera simultánea, por lo tanto la velocidad de transferencia es rápida, sin embargo tiene la desventaja de utilizar una gran cantidad de líneas, por lo tanto se vuelve más costoso y tiene la desventaja de atenuarse a grandes distancias, por la capacitancia entre conductores así como sus parámetros distribuidos.

Existen dos tipos de comunicaciones seriales; la síncrona y la asíncrona. En la comunicación Serial síncrona, se necesitan 2 líneas, una línea sobre la cual se transmitirán los datos y otra la cual contendrá los pulsos de reloj que indicaran cuando

2.2.5.2. Criterios de selección de acelerómetros

Para elegir un sensor de aceleración, además de los márgenes de valores de la aceleración que admite habrá que tener en cuenta si es capaz de medir en continua o sólo en alterna, la máxima frecuencia a la que puede trabajar, así como los correspondientes parámetros instrumentales típicos de todo sensor. (Monje Centeno, 2010)

En la siguiente tabla se resumen algunas de las principales características de los acelerómetros y sus aplicaciones más típicas teniendo en cuenta que el margen de medida se expresa en unidades de g (aceleración de la gravedad terrestre cuyo valor es aproximadamente de 9,81 m/s2).

Tabla 1.Características de los acelerómetros y sus aplicaciones

Tipo	Margen medida(g)	Bw(Hz)	Ventajas e Inconvenientes	Aplicaciones
Micro mecánicos	1.5 – 250	0.1 – 1500	• Alta sensibilidad • Coste medio • Uso sencillo • Bajas temperaturas	• Impacto • ABS • Airbag • Automoción
Piezoeléctricos	0 – 2000	10 – 20000	• Sensibilidad media • Uso complejo • Bajas temperaturas • No funcionan en continua	• Vibración • Impacto • Uso industrial
Capacitivos	0 – 1000	0 – 2000	• Funciona en continua • Bajo ruido • Baja potencia • Excelentes características	• Uso general • Uso industrial
Mecánicos	0 – 200	0 – 1000	• Alta precisión en continua • Lentos • Alto coste	• Navegación inercial • Guía de misiles • Nivelación

En cualquier caso, la selección del acelerómetro para una aplicación concreta se hará en función de una serie de criterios:

- Frecuencia de trabajo o margen de frecuencias de uso, tanto los valores mínimos como los máximos que determinan la velocidad de respuesta que precisamos.
- Los valores máximos y mínimos del nivel de la señal que esperamos. El valor mínimo de señal no suele ser muy importante excepto para algunas aplicaciones en concreto en que se precisan medidas de vibraciones muy débiles. En estos casos, el valor de la señal debería ser de, al menos, cinco veces el valor del ruido generado. Por el contrario, el valor máximo no debería sobrecargar el sensor ni introducir distorsiones en la señal (tanto el sensor como el amplificador asociado podrían salirse de la zona lineal).
- Consideraciones acerca de la forma de montaje, el espacio disponible, la forma de salida de los cables, etc. pueden parecer triviales al usuario, pero pueden hacer que una selección quede invalidada.
- Otras consideraciones tales como la temperatura de trabajo, aspectos ambientales y de compatibilidad química o la necesidad de seguridad intrínseca deberán tenerse en cuenta a la hora de efectuar la elección final del sensor.

2.2.6. Microntroladores

Los microcontroladores son computadores digitales integrados en un chip que cuentan con un microprocesador o unidad de procesamiento central (CPU), una memoria para almacenar el programa, una memoria para almacenar datos y puertos de entrada salida. A diferencia de los microprocesadores de propósito general, como los que se usan en los computadores PC, los microcontroladores son unidades autosuficientes y más económicas. El funcionamiento de los microcontroladores está determinado por el programa almacenado en su memoria.

Este puede escribirse en distintos leguajes de programación. Además, la mayoría de los microcontroladores actuales pueden reprogramarse repetidas veces. Por las características mencionadas y su alta flexibilidad, los microcontroladores son ampliamente utilizados como el cerebro de una gran variedad de sistemas embebidos que controlan maquinas, componentes de sistemas complejos, como aplicaciones industriales de automatización y robótica, domótica, equipos médicos, sistemas aeroespaciales, e incluso dispositivos de la vida diaria como automóviles, hornos de microondas, teléfonos y televisores.

2.2.6.1. Recursos de los microcontroladores

Al estar todos los microcontroladores integrados en un chip, su estructura fundamentalmente y sus características generales son muy parecidas. Todos deben disponer de los bloques esenciales como los son el Procesador (CPU), memoria de datos y de instrucciones, líneas de I/O, oscilador de reloj y módulos controladores de periféricos.

2.2.6.1.1. Arquitectura Básica.

Inicialmente todos los microcontroladores adoptaron la arquitectura clásica de Von Neumann, pero en la actualidad se impone la arquitectura Harvard.

La arquitectura de Von Neumann se caracteriza por disponer de una sola memoria principal donde se almacenan datos e instrucciones de forma indistinta. A dicha memoria se accede a través de un sistema de buses único (direcciones, datos y control) como se observa en la figura 2.8

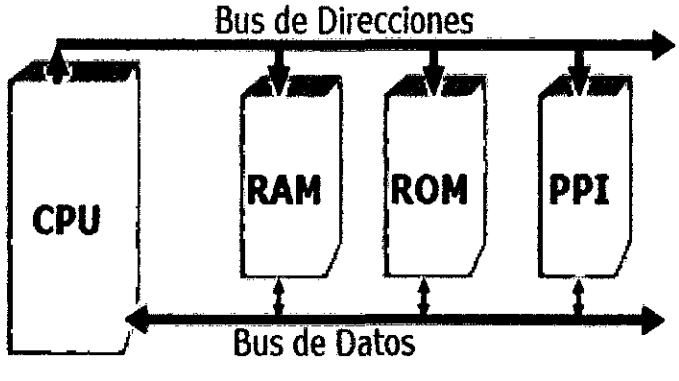


Figura 2. 8. Arquitectura Von Neumann

La arquitectura de Harvard dispone de dos memorias independientes, una que contiene solo instrucciones y otra, solo datos. Ambas disponen de sus respectivos sistemas de buses de acceso y es posible realizar operaciones de acceso (lectura o escritura) simultáneamente en ambas memorias. Los microcontroladores PIC responden a la arquitectura Harvard, como se muestra en la figura 2.9.

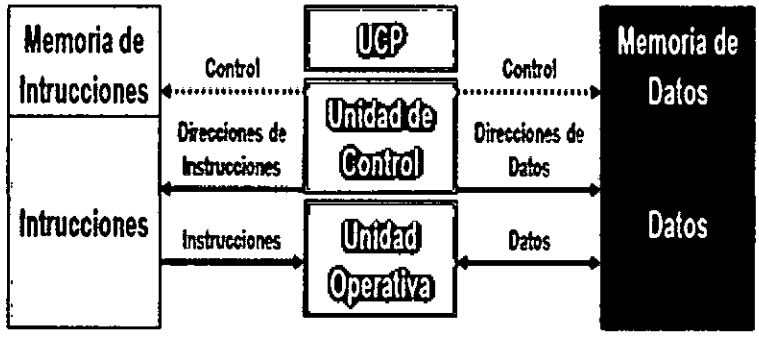


Figura 2. 9 Arquitectura Harvard

2.2.6.1.2. Procesador o CPU

Es el elemento más importante del microcontrolador y determina sus principales características, tanto a nivel de hardware como de software. Existen tres orientaciones en cuanto a la arquitectura y funcionalidad de los procesadores actuales:

- **CISC:** Computadores de Juego de Instrucciones Complejo. Disponen de más de 80 instrucciones maquina en su repertorio, algunas de las cuales son muy sofisticadas y potentes, requiriendo muchos más ciclos para su ejecución. Una ventaja de los procesadores CISC es que ofrecen al programador instrucciones complejas que actúan como macros.
- **RISC:** Computadores de Juego de Instrucciones Reducido. En estos procesadores el repertorio de instrucciones maquina es muy reducido y las

instrucciones son simples y generalmente, se ejecutan en un ciclo. La sencillez y rapidez de las instrucciones permiten optimizar el hardware y el software del procesador.

- **SISC:** Computadores de Juego de Instrucciones Especifico. En los microcontroladores destinados a aplicaciones muy concretas, el juego de instrucciones, además de ser reducido, es "especifico", o sea, las instrucciones se adaptan a las necesidades de la aplicación prevista.

2.2.6.1.3. Memoria

En los microcontroladores la memoria de instrucciones y datos está integrada en el propio chip. Una parte debe ser no volátil, tipo ROM, y se destina a contener el programa de instrucciones que gobierna la aplicación. Otra parte de memoria será tipo RAM, volátil, y se destina a guardar variables y datos.

Se describen las cinco versiones de memoria no volátil que se pueden encontrar en los microcontroladores del mercado:

- **ROM con mascara:** Es una memoria no volátil de solo lectura cuyo contenido se graba durante la fabricación del chip.
- **OTP:** El microcontrolador contiene una memoria no volátil de solo lectura "programable una sola vez" por el usuario. OTP (One Time Programmable).
- **EPROM:** Los microcontroladores que disponen de memoria EEPROM (Erasable Programmable Read Only Memory) pueden borrarse y grabarse muchas veces.
- **EEPROM:** Se trata de memorias de solo lectura, programables y borrables eléctricamente EEPROM

(Electrical Erasable Programmable Read Only Memory).

- **FLASH:** Se trata de una memoria no volátil, de bajo consumo, que se puede escribir y borrar. Funciona como una ROM y una RAM pero consume menos y es más pequeña.

2.2.6.1.4. Puertas de entrada y salida (I/O)

La principal utilidad de los pines que contiene un microcontrolador es soportar las líneas de I/O que comunican al computador interno con los periféricos exteriores.

Según los controladores de periféricos que posea cada modelo de microcontrolador, las líneas de I/O se destinan a proporcionar el soporte a las señales de entrada, salida y control. Por lo general estas líneas se agrupan de ocho en ocho formando puertos.

2.2.6.1.5. Reloj principal

Todos los microcontroladores disponen de un circuito oscilador que genera una onda cuadrada de alta frecuencia, que configura los impulsos de reloj usados en la sincronización de todas las operaciones del sistema.

Aumentar la frecuencia de reloj supone disminuir el tiempo en que se ejecutan las instrucciones pero lleva aparejado un incremento en el consumo de energía.

2.2.6.1.6. Temporizadores o Timers

Generalmente son registros de función especial SRF de 8 o 16 bits los cuales son empleados para controlar periodos de tiempo cuando utilizan el oscilador de cuarzo interno para su funcionamiento permitiendo medir el tiempo entre dos eventos (temporizadores), o para realizar el conteo de eventos que suceden en el exterior si la fuente que genera los pulsos es externa (contador). Una vez que se llena el registro, se genera una interrupción por desbordamiento.

2.2.6.1.7. Perro guardián o "Watchdog"

Consiste en un temporizador que, cuando se desborda y pasa por cero, provoca un reset automáticamente en el sistema. Se debe diseñar el programa de trabajo que controla la tarea de forma que refresque o inicialice el Perro guardián antes de que provoque el reset. Si falla el programa o se bloquea, no se refrescara el Perro guardián y, al completar la temporización, "ladrara y ladrara" hasta provocar el reset.

2.2.6.1.8. Protección ante fallo de alimentación o "Brownout"

Circuito que resetea al microcontrolador cuando el voltaje de alimentación (VDD) es inferior a un voltaje mínimo (Brownout).

2.2.6.1.9. Estado de reposo o bajo consumo

Los microcontroladores disponen de una instrucción especial (SLEEP en los PIC), que les pasa al estado de reposo o de bajo consumo, en el cual los requerimientos de potencia son mínimos.

2.2.6.2. Características especificas

2.2.6.2.1. Conversor A/D

Los microcontroladores que incorporan un Conversor A/D (Analógico / Digital) están en la capacidad de convertir señales analógicas que provienen del mundo real en números digitales discretos.

2.2.6.2.2. Conversor D/A

Transforma los datos digitales obtenidos del procesamiento del computador en su correspondiente señal analégica que saca al exterior por una de las patillas del encapsulado.

2.2.6.2.3. Comparador analógico

Algunas modelos de microcontroladores disponen internamente de un amplificador operacional que actúa como comparador entre una señal fija de referencia y otra variable que se aplica por una de los pines del encapsulado. La salida del comparador proporciona un nivel lógico de 1 ó 0 según una señal sea mayor o menor que otra.

2.2.6.2.4. Modulador por ancho de pulso o PWM

Circuitos que proporcionan en su salida impulsos de ancho variable a una frecuencia determinada, los cuales se ofrecen al exterior a través de pines.

2.2.6.2.5. Puertos de comunicación

Con el objetivo de dotar al microcontrolador de la posibilidad de comunicarse con otros dispositivos externos, otros buses de microprocesadores, buses de sistemas, buses de redes y poder adaptarlos con otros elementos bajo otras normas y protocolos. Algunos modelos disponen de recursos que permiten directamente esa tarea, entre los que destacan:

- **UART:** adaptador de comunicación serie asíncrona.
- **USART:** adaptador de comunicación serie síncrona y asíncrona.
- **USB (universal serial bus):** moderno bus serie para los PC.
- **I^2C:** bus que consiste en una interfaz de dos hilos desarrollada por Philips.
- **CAN (Controller Área Network):** diseñado para permitir la adaptación con redes de conexionado multiplexado se desarrolló conjuntamente por Bosch e Intel.

2.2.7. Microcontroladores PIC

Para lograr una compactación de código óptima y una velocidad superior a la de sus competidores los microcontroladores PIC de Microchip incorporan en su procesador tres características más avanzadas en los grandes computadores:

- Procesador **RISC**
- Procesador **Segmentado**
- Arquitectura **Harvard**

Con la incorporación de estos recursos los PIC son capaces de ejecutar en **un ciclo** de instrucción todas las instrucciones, excepto las del salto, que tardan el doble. Otra característica relevante de los PICS es el manejo intensivo del **Banco de Registros,** los cuales participan de una manera muy activa en la ejecución de las instrucciones, lo cual implica que todos los elementos del sistema, es decir, temporizadores, puertos de entrada/salida, posiciones de memoria, etc., están implementados físicamente como registros.

Por ejemplo la **ALU** (unidad aritmética-lógica) efectúa sus operaciones lógico-aritméticas con dos operandos, uno que es el que recibe desde el registro **W** (work), que hace que las veces de acumulador en los microprocesadores convencionales, y otro que puede provenir de cualquier registro o del propio código de la instrucción. El resultado de la operación puede almacenarse en cualquier registro o en **W** como se muestra en la figura 2.10. Esta funcionalidad da un carácter totalmente ortogonal a las instrucciones que puede utilizar cualquier registro como **operando fuente y destino.** La memoria RAM implementa en sus posiciones los registros específicos y los de propósito general.

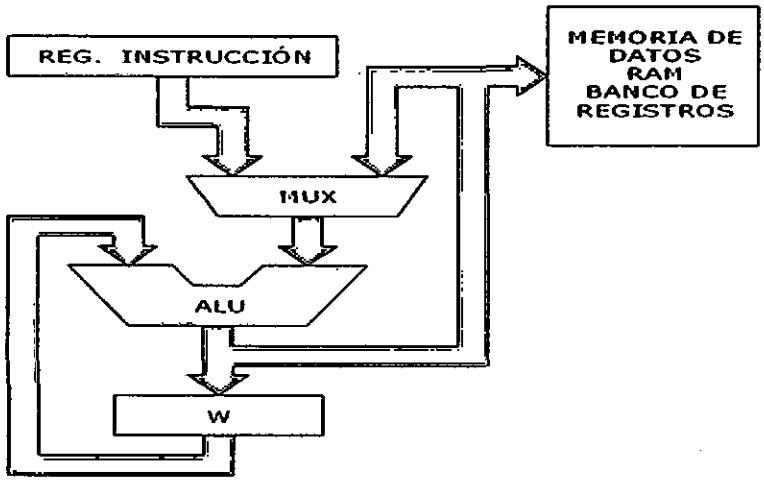


Figura 2. 10 Modo de operación de la ALU

2.2.7.1. Clasificación de los microcontroladores PIC

Los microcontroladores se clasifican de acuerdo al tamaño de la data que pueden manejar, estos pueden ser de 8, 16 y 32 bits.

2.2.7.1.1. Pic de 8 bits

La longitud de los datos que maneja es de 8 bits, esto corresponde al tamaño del bus de datos y el de los registros de la CPU. Los PIC de 8 bits se clasifican en 3: gama base, gama media y gama mejorada.

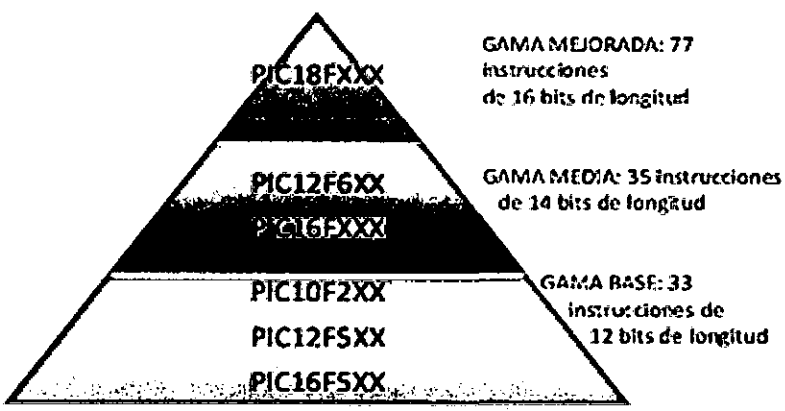


Figura 2. 11 Clasificación de los PIC de 8 bits

- **Gama base**

  Estos modelos responden a 33 instrucciones máquina de 12 bits de longitud cada una y disponen de una pila con 2 niveles de profundidad. Su capacidad de memoria es muy limitado por lo que las aplicaciones que se pueden hacer con el también. Lo componen 14 modelos, 6 tienen 6 patitas y son llamados enanos, a continuación una gráfica para ver como aumenta la capacidad de memoria del PIC con el número de patitas.

  Los PIC de 6 patitas han tenido un gran éxito debido a su bajo costo, volumen y que se resuelven bastantes aplicaciones simples con ellos. En este caso la alimentación se aplica a dos de las patitas y quedan 4 restantes para las entradas y salidas y las funciones de sus periféricos, como un Timer (temporizador), comparador analógico, un ADC de 8 bits, etc. Favoreciendo la migrabilidad de hardware y portabilidad de código, cuando se cambia a un PIC con pines de 8, 14 y 20, las patitas siguen teniendo la misma función, de esta forma los cambios serán mínimos

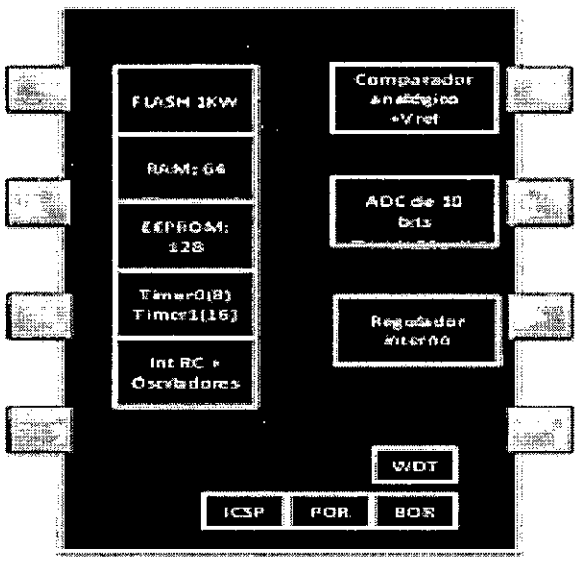


Figura 2. 12 Características de los PIC de gama base

- **Gama media**

Estos pics responden a 35 instrucciones con 14 bits de longitud cada una, tiene una pila de 8 niveles de profundidad y tiene un vector de corrupción. Esta gama tiene en la actualidad 71 modelos diferentes, tienen de 8 a 68 patitas.

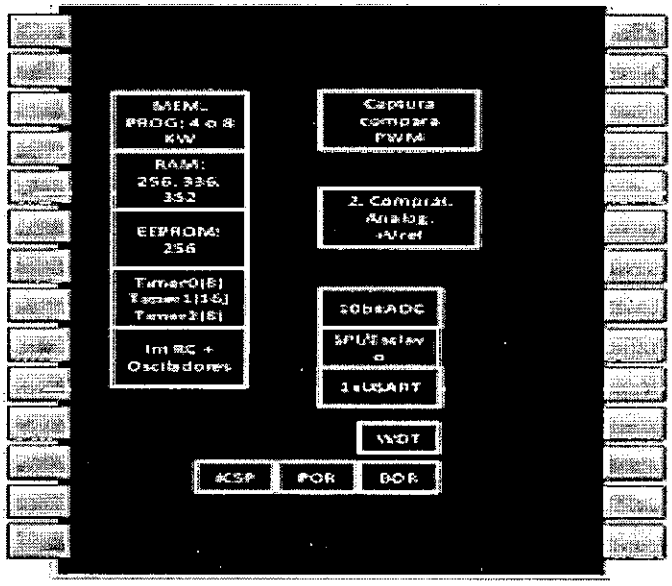


Figura 2. 13 Características de los PIC de gama media

- **Gama mejorada**

Estos PIC disponen de 77 instrucciones de 16 bits de longitud cada una, una Pila de 31 niveles de profundidad y 2 vectores de interrupción. Poseen la nomenclatura PIC18Xxxx, que se muestra a continuación en la figura 2.14.

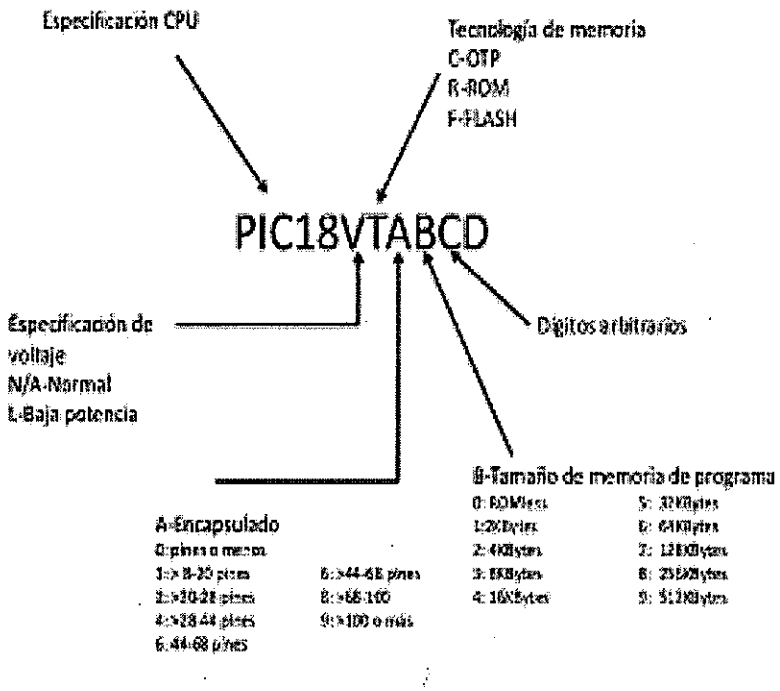


Figura 2. 14 Nomenclatura de los PIC de la serie 18

La capacidad de memoria de programa es de 128 KB máximo, la de datos es de 3963 bits y la de EEPROM de hasta 1 KB. Tiene periféricos muy especializados como un Conversor AD, 5 temporizadores, interfaces para comunicación con bus, etc. Entre las características a destacar es un multiplicador rápido hardware que permite hacer una operación en un ciclo de una instrucción. Tiene de 68 a 80 patitas.

2.2.7.1.2. Pics de 16 bits

Actualmente las aplicaciones necesitan microcontroladores más potentes que los de 8 bits, es por eso que se crearon los de 16 bits. Existen 2 grandes gamas de microcontroladores de 16 bits:

1) La gama MCU, que está conformada por las familias de dispositivos PIC24F y PIC24H.

2) La gama DSC de 16 bits, formada por las familias dsPIC30F y dsPIC33F.

- **Gama de microcontroladores MCU de 16 bits**

Los modelos con nomenclatura PIC24F pertenecen a esta gama, se fabrican con una tecnología de 0.25 micras, tienen un rendimiento de 16 MIPS a 32 MHz y está orientada a solucionar diseños que no se podía con PIC18Xxxx.

Los PIC24F tienen una arquitectura Harvard modificada con un bus de datos de 16 bits y con instrucciones de longitud 24 bits. Manejan una memoria de programa lineal de hasta 8 MB y una de datos de hasta 64 KB.
El núcleo del procesador se alimenta con 2.5 V, las líneas de entrada/salida con 3.3 V.

- **Gama de microcontroladores DSC de 16 bits**

Los microcontroladores de 16 bits tienen las características típicas de los MCU y características propias de los DSP (Procesadores Digitales de Señales) por lo que se pueden usar en aplicaciones relacionadas con el proceso digital de señales. Aprovechan las ventajas de los microcontroladores MCU y DSP.

Los DSC cuentan con importantes capacidades de memoria, contienen la mayoría de los recursos y periféricos y tienen un rendimiento de 30 MIPS cuando es alimentado con voltaje de 4.5 y 5.5 V, hay modelos con 256 KB de memoria FLASH y 30 KB de RAM. A esta gama pertenecen los dsPIC33F, de estos existe un total de 27 dispositivos, 15 de estos son de propósito general y el resto el de Control de Motores.

Los modelos dsPIC33F de propósito general son adecuados para aplicaciones de voz y audio, seguridad, electro medicina, módem, etc. Los de control de motores están orientados a los electrodomésticos como las lavadoras, al control de la dirección electrónica de automóviles, control medioambiental, sistemas de alimentación ininterrumpible, inversores y a matrices de iluminación LED.

2.2.8. Comandos AT

La mayoría de los módems se controlan y responden a caracteres enviados desde la computadora. El lenguaje de comandos para modem más extendido es el de los comandos Hayes que fue inicialmente incorporado a los módems de este fabricante.

Casi todos estos comandos comienzan con **AT (Attention)** y terminan con un retorno del carro (enter), por lo primero es que son comúnmente llamados comandos AT.

Cada modem utiliza una serie de órdenes "AT" comunes y otras específicas. El uso exacto del término comandos AT varía levemente de fabricante a fabricante.

En general se puede asumir que un modem con un conjunto de comandos AT usa la forma original de Hayes de separar datos y comandos, soporta los comandos y registros (de configuración) originales de Hayes como un subconjunto.

Aunque la finalidad principal de los comandos AT es la comunicación con módems, la telefonía móvil GSM también ha adoptado como estándar este lenguaje para poder comunicarse con sus terminales. De esta forma, todos los teléfonos móviles GSM poseen un juego de comandos AT específico que sirve de interfaz para configurar y proporcionar instrucciones a los terminales. Este juego de instrucciones puede encontrarse en la documentación técnica de los

terminales GSM y permite acciones tales como realizar llamadas de datos o de voz, leer y escribir en la agenda de contactos y enviar mensajes SMS, además de muchas otras opciones de configuración del terminal.

Queda claro que la implementación de los comandos AT corre a cuenta del dispositivo GSM y no depende del canal de comunicación a través del cual estos comandos sean enviados, ya sea cable de serie, canal Infrarrojos, Bluetooth, etc. De esta forma, es posible distinguir distintos teléfonos móviles del mercado que permiten la ejecución total del juego de comandos AT o sólo parcialmente. Por ejemplo, Nokia 6820 no permite la ejecución de comandos AT relativos al manejo de la memoria de agenda de contactos y llamadas pero sí que permite acceder al servicio SMS; Nokia 6600 (basado en Symbian) no permite la ejecución de comandos AT relativos a la gestión de la agenda ni de SMSs.

2.2.9. Código ASCII

El código ASCII (siglas en inglés para ***American Standard Code for Information Interchange***), es decir Código Americano Estándar para el intercambio de Información.

Fue creado en 1963 por el Comité Estadounidense de Estándares o "ASA", este organismo cambio su nombre en 1969 por "Instituto Estadounidense de Estándares Nacionales" o "ANSI" como se lo conoce desde entonces.

Este código nació a partir de reordenar y expandir el conjunto de símbolos y caracteres ya utilizados en aquel momento en telegrafía por la compañía Bell. En un primer momento solo incluía letras mayúsculas y números, pero en 1967 se agregaron las letras minúsculas y algunos caracteres de control, formando así lo que se conoce como US-ASCII, es decir los caracteres del 0 al 127. Así con este conjunto de solo 128 caracteres fue publicado en 1967 como estándar, conteniendo todos lo necesario para escribir en idioma inglés.

En 1981, la empresa IBM desarrolló una extensión de 8 bits del código ASCII, llamada "página de código 437", en esta versión se reemplazaron algunos caracteres de control obsoletos, por caracteres gráficos. Además se incorporaron 128 caracteres nuevos, con símbolos, signos, gráficos adicionales y letras latinas, necesarias para la escrituras de textos en otros idiomas, como por ejemplo el español. Así fue como se sumaron los caracteres que van del ASCII 128 al 255.

IBM incluyó soporte a esta página de código en el hardware de su modelo 5150, conocido como "IBM-PC", considerada la primera computadora personal. El sistema operativo de este modelo, el "MS-DOS" también utilizaba el código ASCII extendido.

Casi todos los sistemas informáticos de la actualidad utilizan el código ASCII para representar caracteres, símbolos, signos o una extensión compatible para representar textos y para el control de dispositivos que manejan texto como el teclado. No deben confundirse los códigos **ALT+ número** de teclado con los códigos **ASCII**.

2.2.9.1. Caracteres de control ASCII

El código ASCII reserva los primeros 32 códigos (numerados del 0 al 31 en decimal) para caracteres de control: códigos no pensados originalmente para representar información imprimible, sino para controlar dispositivos (como impresoras) que usaban ASCII. Por ejemplo, el carácter 10 representa la función "nueva línea" (**line feed**), que hace que una impresora avance el papel, y el carácter 27 representa la tecla **"escape"** que a menudo se encuentra en la esquina superior izquierda de los teclados comunes. El código 127 (los siete bits a uno), otro carácter especial, equivale a "suprimir" **("delete").** Dado que el código 0 era ignorado, fue posible dejar huecos (regiones de agujeros) y más tarde hacer correcciones.

Muchos de los caracteres de control ASCII servían para marcar paquetes de datos, o para controlar protocolos de transmisión de datos (por ejemplo **ENQuiry**, con el significado: ¿hay alguna estación por

ahí?, **ACKnowledge**: recibido o ", **Start Of Header**: inicio de cabecera, **Start of Text**: inicio de texto, **End of Text**: final de texto, etc.). **ESCape** y **SUBstitute** permitían a un protocolo de comunicaciones, por ejemplo, marcar datos binarios para que contuviesen códigos con el mismo código que el carácter de protocolo, y que el receptor pudiese interpretarlos como datos en lugar de como caracteres propios del protocolo.

Los diseñadores del código ASCII idearon los caracteres de separación para su uso en sistemas de cintas magnéticas. Dos de los caracteres de control de dispositivos, comúnmente llamados XON y XOFF generalmente ejercían funciones de caracteres de control de flujo para controlar el flujo hacia un dispositivo lento (como una impresora) desde un dispositivo rápido (como una computadora), de forma que los datos no saturasen la capacidad de recepción del dispositivo lento y se perdiesen.

Los primeros usuarios de ASCII adoptaron algunos de los códigos de control para representar "meta-información" como final-de-línea, principio/final de un elemento de datos, etc. Estas asignaciones a menudo entraban en conflicto, así que parte del esfuerzo de convertir datos de un formato a otro comporta hacer las conversiones correctas de meta-información. Por ejemplo, el carácter que representa el final-de-línea en ficheros de texto varía con el sistema operativo.

Cuando se copian archivos de un sistema a otro, el sistema de conversión debe reconocer estos caracteres como marcas de final-de-línea y actuar en consecuencia. Actualmente los usuarios de ASCII usan menos los caracteres de control, (con algunas excepciones como "retorno de carro" o "nueva línea").

un dato es válido. Ejemplos: de este tipo de comunicación son los protocolos I2C (Inter Integrated Circuit) y SPI (Serial Peripherical Interface).

En la comunicación **Serial asíncrona**, no son necesarios los pulsos de reloj. La duración de cada bit está determinada por la velocidad con la cual se realiza la transferencia de datos. La figura 3.1 muestra la estructura de un carácter que se trasmite en forma serial asíncrono. Normalmente cuando no se realiza ninguna transferencia de datos, la línea del transmisor se encuentra en estado de **Idle,** esto quiere decir en un estado alto.

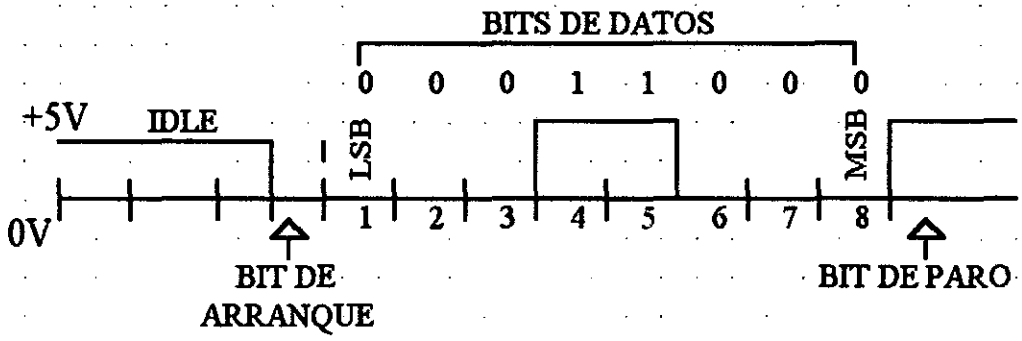


Figura 3. 1 Secuencia de Transmisión serial asíncrona

Para iniciar la transmisión de datos, el transmisor coloca esta línea en bajo durante un determinado tiempo, a lo cual se le conoce como bit de arranque (**Start bit**) y a continuación empieza a transmitir en un intervalo de tiempo fijo, los bits correspondientes al dato, empezando siempre por el BIT menos significativo (**LSB**), y terminando con el BIT más significativo. Si el receptor no está sincronizado con el transmisor, este desconoce cuándo se van a recibir los datos. Por lo tanto el transmisor y el receptor deberán tener los mismos parámetros de velocidad, paridad, número de bits del dato transmitido y de BIT de parada.

En circuitos digitales, cuyas distancias son relativamente cortas, se puede manejar transmisiones en niveles lógicos TTL (0-5V), pero cuando las distancias aumentan, estas señales tienden a distorsionarse debido al efecto capacitivo de los conductores y su resistencia eléctrica. El efecto se incrementa a medida que se incrementa la velocidad de la transmisión. Todo esto origina que los datos recibidos no sean igual a los datos transmitidos, por lo que no se puede permitir la transferencia de datos.

Ante la gran variedad de equipos, sistemas y protocolos que existen surgió la necesidad de un acuerdo que permitiera a los equipos de varios fabricantes comunicarse entre sí. La **EIA** *(Electronics Industry Association)* por los años 60 elaboro la norma RS-232, la cual define la interface mecánica, los pines, las señales y los protocolos que debe cumplir la comunicación serial para comunicar un equipo terminal de datos o DTE (*Data Terminal Equipment*, el PC en este caso) y un equipo de comunicación de datos o DCE (*Data Communication Equipment*), ubicado a distancias no mayores a 15 metros (aunque en la práctica alcanza distancias de hasta 50 metros) y a una velocidad máxima de 19,200 bps.

Este tipo de trasmisión se le conoce como ***"single ended"*** o no balanceada porque usa en el cable un solo retorno (GND). Es un modo de trasmisión muy simple, pero también vulnerable al ruido aditivo en la línea y por esa razón es empleada para comunicación a distancias cortas.

El formato de trasmisión de datos en las señales TX y RX del estándar RS232, se muestra en la figura 3.2. Se trata de una señal serial bipolar, normalmente entre +10 y -10 volts, con formato asíncrono. En el ejemplo se trasmite el código ASCII de la "A" (01000001). Observe que sigue una lógica negativa, con un nivel alto para el valor 0 lógico y un nivel bajo para el 1. En la línea de tiempo, que va de izquierda a derecha, el bit menos significativo LSB se trasmite primero y el bit más significativo MSB, al último.

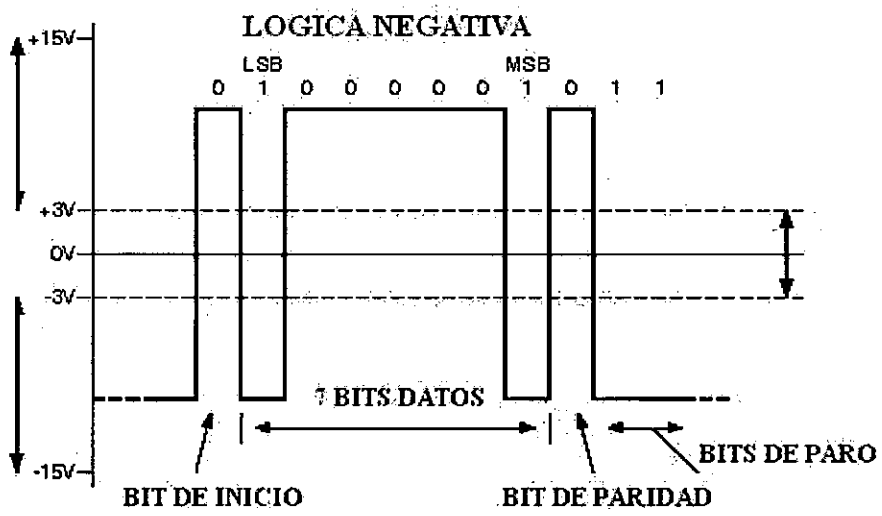


Figura 3. 2 Formato de transmisión para la norma RS232

Todas las normas RS-232 cumplen con los siguientes niveles de voltaje:

- Un "1" lógico es un voltaje comprendido entre –5v y –15v en el transmisor y entre -3v y –25v en el receptor.
- Un "0" lógico es un voltaje comprendido entre +5v y +15 v en el trasmisor y entre +3v y +25 v en el receptor.
- La región entre +3v y -3v es indefinida.
- Un circuito abierto nunca debe exceder los 25v (con referencia a GND)
- La corriente de corto circuito nunca debe exceder a los 500mA. El manejador debe ser capaz de manejar esa corriente sin dañarse.

Para la conexión se utiliza un cable con conectores DB9, con nueve señales, como el mostrado en las figura 3.3. Adicionalmente a la señales de datos trasmitidos y recibidos TX, RX, la norma original RS232 incluye definiciones para señales de control (en inglés "handshake signals") que se usan para varias funciones auxiliares en el protocolo de envío y recepción de datos, así como para el diagnóstico de fallas.

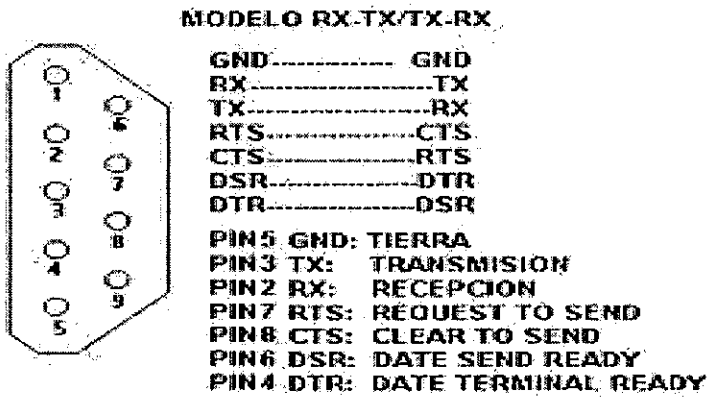


Figura 3. 3 Descripción de pines del conector DB9

Actualmente, el estándar se ha simplificado a las señales de trasmisión TX, recepción RX y tierra GND, dejando sin utilizarse el resto de las señales. La aplicación más común es la comunicación entre una computadora y una terminal remota.

El envío de niveles lógicos (bits) a través de cables o líneas de transmisión necesita la conversión a voltajes apropiados. En los microcontroladores para representar un **'0 lógico'** se trabaja con voltajes inferiores a 0.8V y para un **'1 lógico'** con voltajes

mayores a 2V. En general cuando se trabaja con familias TTL y CMOS se asume que un "0" lógico es igual a cero Volts y un "1" lógico es igual a 5 Volts.

La importancia de conocer esta norma, radica en los niveles de voltaje que maneja el puerto serial del ordenador, ya que son diferentes a los que utilizan los microcontroladores y los demás circuitos integrados. Por lo tanto se necesita de una interface que haga posible la conversión de niveles de voltaje a los estándares manejados por los CI TTL, donde por lo general se utiliza un circuito basado en la MAX232 o convertidor USB-Serial, que permiten conseguir la adaptación de niveles necesaria para trabajar con microcontroladores o dispositivos TTL.

3.3 Protocolo de comunicación I^2C

El bus I2C fue desarrollado por Philips al inicio de la década de 1980, teniendo en mente todos los siguientes aspectos:

- **Bus serie síncrono**. El término *síncrono* deberá entenderse como que existirá, además de la línea de datos, una señal de sincronía explícita para validar los datos en el canal serie, y no en el sentido de que en la línea de datos siempre deberá existir un flujo de datos, bien propiamente como tales o bien como medio para mantener la sincronía.
- **El sentido del enlace será semibidireccional**. Es decir, existirá una única línea de datos que podrá utilizarse para el flujo en ambos sentidos, pero no simultáneamente. De esta manera se ahorra el número de señales del bus. Esta limitación es un hecho menor debido al ámbito de aplicación de este tipo de bus, en el que no se necesita la bidireccionalidad.
- **Deberá admitir topologías multipunto**. Es decir, podrán conectarse al bus varios dispositivos, pudiendo actuar varios de ellos como emisores, aunque no simultáneamente.
- **Un mismo dispositivo podrá actuar o como emisor o como receptor en distintos momentos.**
- **Deberá admitir topologías multi-maestro**. Es decir, más de un dispositivo puede intentar gobernar el bus. Un maestro podrá actuar tanto como emisor como receptor; y lo mismo puede decirse de un esclavo. Lo que diferenciará a un

maestro de un esclavo es la capacidad de gobierno del bus (suministro de la señal de sincronía).

- **Deberá establecer un mecanismo de arbitraje.** Para el caso en que simultáneamente más de un maestro intente gobernar el bus.
- **Un maestro podrá funcionar también como un esclavo**. El motivo es que en las topologías multi-maestro un maestro que haya ganado el gobierno del bus pueda dirigirse a cualquier otro maestro, que entonces deberá comportarse como un esclavo.
- **Deberá establecer un mecanismo de adaptación de velocidad**. Para el caso en que un esclavo no pueda seguir la velocidad impuesta por el maestro.
- **La velocidad de las transferencias, marcada por el maestro, podrá ser variable dentro de unos límites**. Además, por su ámbito de aplicación, la tasa de transferencia máxima podrá ser relativamente pequeña, entre 100 y 400 kilobits por segundo normalmente.
- **Deberá establecer un criterio de *direccionamiento* abierto.** Mediante el cual se sepa a qué esclavo se dirige un maestro. Además, la *dirección* asignada a un circuito podrá ser modificada dinámicamente (no confundir el concepto de *direccionamiento de sistema* con el de direccionamiento de una memoria). Por otra parte, el mecanismo de *direccionamiento* deberá admitir futuras ampliaciones del número máximo de elementos direccionables.
- **El formato de las tramas transmitidas habrá de ser suficientemente flexible.** Con el objetivo de poder adaptarse a la funcionalidad de cualquier tipo de circuito pasado, presente o futuro.
- **Desde el punto de vista de la capa física de la interfaz, deberá admitir la conexión de circuitos de diferentes tecnologías** (diversas bipolares, MOS, etcétera).

El bus I2C sólo define dos señales, además del común:

- **SDA**. Es la línea de datos serie (***Serial DAta***), semibidireccional. Eléctricamente se trata de una señal a colector o drenador abierto. Es gobernada por el emisor, sea éste un maestro o un esclavo.

- **SCL**. Es la señal de sincronía (reloj serie, o *Serial CLock* en inglés). Eléctricamente se trata de una señal a colector o drenador abierto. En un esclavo se trata de una entrada, mientras que en un maestro es una salida.

Un maestro, además de generar la señal de sincronía suele tener la capacidad de evaluar su estado; el motivo –como se verá– es poder implementar un mecanismo de adaptación de velocidades. Esta señal es gobernada única y exclusivamente por el maestro; un esclavo sólo puede *retenerla* o *pisarla* para forzar al maestro a ralentizar su funcionamiento, cosa que se explicará más adelante. Esta particularidad física de que las salidas de los excitadores I2C hayan de ser a colector o drenador abierto no es casual sino que resulta de vital importancia para que a este bus con tan sólo una señal de datos y otra de sincronía se le pueda dotar de todas las características funcionales apuntadas en el apartado anterior. Recuérdese que un dispositivo lógico con este tipo de salida permite realizar la función producto lógico con una simple conexión eléctrica (montaje *Y por conexión*).

Evidentemente, un enlace I2C necesitará sendas resistencias de elevación en las respectivas líneas SDA y SCL. De esta manera un excitador I2C realmente sólo gobierna el estado 0 lógico en las líneas I2C, mientras que el estado 1 lógico no es suministrado por el excitador directamente sino por medio de la oportuna resistencia de pull-up. Así, el concepto de *aislamiento del bus* se consigue con tal sólo hacer que el excitador del nodo que se desea aislar ofrezca a su salida el estado alto (1 lógico), al que le corresponde una elevadísima impedancia de salida, equivalente a un aislamiento eléctrico con respecto al bus.

3.3.1 Estados del bus i2c

El bus I2C puede encontrarse en distintos estados, que la norma define como los siguientes:

- **Libre**. Este estado se caracteriza por encontrase las líneas SDA y SCL a 1, sin que se esté realizando ningún tipo de transferencia.

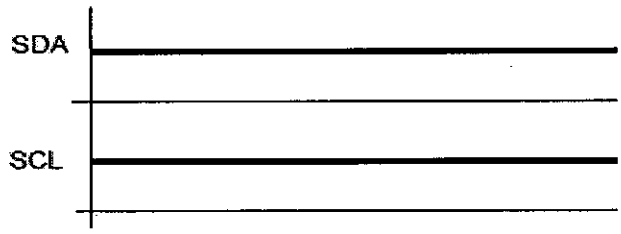


Figura 3. 4 Condición de bus I²C libre

- **Inicio**. Se produce una condición de *inicio* cuando un **maestro** inicia una transacción. En concreto, consiste en un cambio de alta a baja en la línea SDA mientras SCL permanece a alta. Esto puede apreciarse en el cronograma mostrado en la figura 3.5. A partir de que se dé una condición de inicio se considerará que el bus está ocupado y ningún otro maestro deberá intentar generar su condición de inicio.

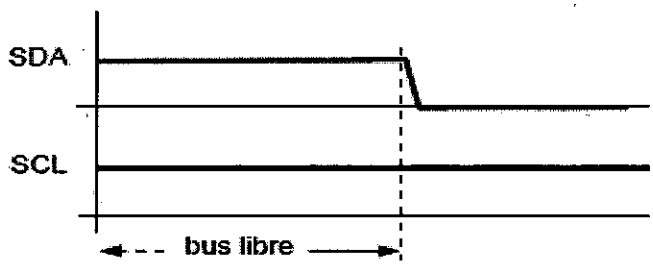


Figura 3. 5 Condición de inicio del bus I²C

- **Cambio**. Se produce una condición de *cambio* cuando, estando a baja la línea SCL, la línea SDA puede cambiar de estado. En la transferencia de datos por un bus I2C éste es el único instante en el que el sistema emisor (que podrá ser tanto un maestro como un esclavo) podrá poner en la línea SDA cada bit del carácter a transmitir.

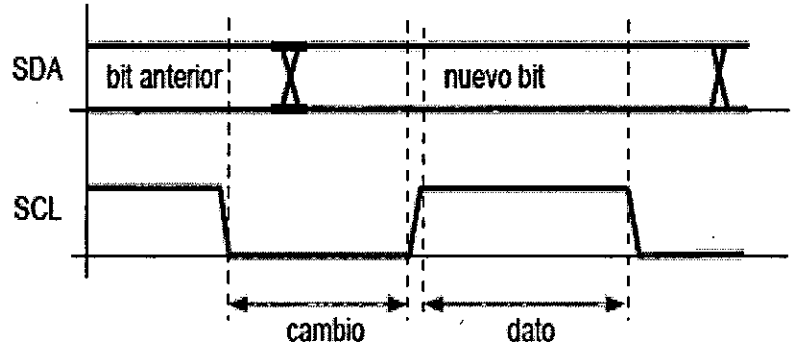


Figura 3. 6 Condiciones o estados de cambio y de dato en el bus I²C

- **Dato**. Este estado es el que, una vez iniciada una transacción, queda definido por la fase alta de la señal de sincronía SCL. En este estado se considera que el dato emitido es válido, y no se admite que pueda cambiar. Recuérdese que, ya iniciada una transferencia, el único instante en que la línea de datos puede cambiar es en el estado de *cambio*.
- **Parada**. Se produce una condición de *fin* o *parada* cuando, estando la línea SCL a alta, se produce un cambio de baja a alta en la línea SDA. Obsérvese que esto es una violación de la condición de *dato*, y es precisamente por esto por lo que se utiliza para que un maestro pueda indicar al esclavo que se finaliza la transferencia. Tras la condición de parada se entra automáticamente en el estado de *bus libre*.

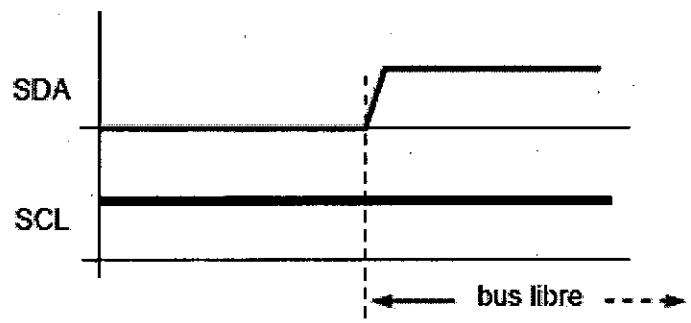


Figura 3. 7 Condición de parada del bus I²C

3.3.2 El formato de una transacción

Entenderemos por *transacción* todos los eventos desarrollados entre una condición de inicio y una de parada. Una vez producido el inicio de una transacción comienza propiamente ésta, y en ella se lleva a cabo la transferencia de uno o varios caracteres, en uno u otro sentido.

- **El carácter de ocho bits es la unidad de transferencia**. El orden de emisión de los bits de un carácter es empezando por el más significativo, siguiendo en orden decreciente de pesos y terminando por el menos significativo (como se ve, es un criterio contrario al que siguen las UART).
- Un carácter puede tener diversos significados en función de quién lo emita y en qué instante lo haga.

- **Tras cada carácter se debe producir un *reconocimiento* por parte del receptor**; el motivo es dotar al protocolo de un mecanismo de seguridad.
- El primer carácter transferido lo emite siempre el maestro; sus siete bits más significativos indican la dirección del esclavo al que se dirige, y el bit de menor peso indica el sentido de la transferencia de los subsiguientes caracteres (0=escritura, 1=lectura, siempre desde el punto de vista del maestro).
- Dentro de una transacción es posible cambiar el sentido del flujo, pero para ello hace falta generar una nueva condición de inicio (*reinicio*) y direccionar de nuevo el mismo esclavo.

Como puede verse, para el direccionamiento de un esclavo se tienen siete bits, lo que daría un máximo de 128 dispositivos I2C distintos. No obstante, en la realidad se dispone de muchas menos direcciones puesto que un cierto número de ellas no se emplean como tales sino como identificadores de función especial, como se verá más adelante; además, ciertos circuitos reservan para sí varias direcciones debido a que es frecuente que en un diseño se tengan que emplear varios del mismo tipo. En estos casos los circuitos tienen una dirección compuesta por dos partes; una fija establecida por el comité I2C y otra variable (bits inferiores) fijada por el diseñador de cada sistema dado. En el encapsulado de estos circuitos existen ciertas patillas (una, dos o tres) mediante las cuales el diseñador puede determinar la parte *programable* de la dirección I2C. Por ejemplo, los circuitos EEPROM tienen asignada una dirección 1010XXX; 1010 es la parte fija de los siete bits, y XXX es la parte programable que corresponde al estado aplicado a tres patillas *ad hoc* de su encapsulado.

Retomando la cuestión de la limitación del número de direcciones, este hecho trajo aparejado el que pocos años después del lanzamiento del bus I2C, y debido a su éxito, se fuesen agotando las direcciones disponibles, lo que implicaba la imposibilidad de ofrecer nuevos circuitos I2C. Afortunadamente, se previó esta contingencia dado que se reservaron para futuras ampliaciones las direcciones que comenzaban por 1111. De esta

manera, ahora **existen dos tipos direcciones: las normales de siete bits y las ampliadas de diez bits**. En este último caso el direccionamiento de un esclavo implica la transferencia de dos caracteres; el primero deberá comenzar por 11110XX, siendo XX los dos bits de mayor peso de la dirección de diez bits, y el segundo carácter corresponderá a los ocho bits inferiores de la dirección del esclavo.

En términos generales, una transacción obedecerá a alguna de las siguientes estructuras:

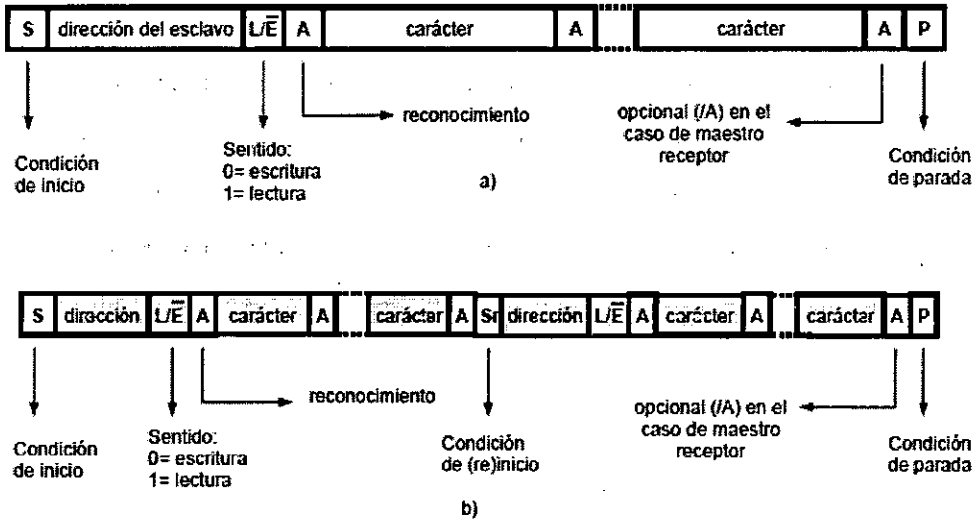


*Figura 3. 8 Formatos de transacción del bus I²C: **a)** sin cambio de sentido **b)** con re direccionamiento y posible cambio de sentido*

3.3.3. El mecanismo de sincronización

Ya se ha comentado que el bus I2C establece un mecanismo para que un esclavo lento pueda forzar a un maestro rápido a ir más lento; a este mecanismo la norma I2C lo denomina *sincronización*. Se trata de un procedimiento ingenioso que saca provecho de la naturaleza a colector o drenador abierto de la señal de reloj SCL. Por defecto, un maestro gobernará la señal de reloj del bus, SCL, a la velocidad que considere oportuno. Tan es así que no tiene por qué ser siempre la misma ni siquiera dentro de una misma transacción ni dentro de un mismo carácter. En este sentido tan sólo se han de satisfacer las condiciones mínimas de temporización entre los instantes de

cambio de las señales, aspectos éstos que más adelante se detallarán al hablar de las especificaciones eléctricas y temporales de la interfaz I2C.

Por lo dicho, el maestro marcha a su propia velocidad y supondrá que el esclavo con el que se comunique será capaz de seguir el ritmo. En muchas circunstancias esto es así, pero en otras puede resultar que no, bien porque el esclavo de ninguna manera pueda seguir jamás tal velocidad marcada por el maestro, o bien porque en ciertos instantes necesite realizar algunos procesos que le consuman un tiempo que no se pueda dedicar a la comunicación con el maestro. Sea cual sea la causa, el esclavo no estará en condiciones de comunicarse correctamente y se generará un error de comunicación. Para dar mayor flexibilidad al bus, éste admite mecanismos para establecer una especie de diálogo entre las partes para amoldar dinámicamente la velocidad.

El mecanismo consiste en lo siguiente: un maestro que admita la *sincronización* deberá comprobar el estado de la línea SCL del bus; si al activarla ésta no se activa efectivamente en el bus entonces querrá significar que el esclavo le está solicitando ralentizar su velocidad. De esta manera, el maestro esperará con su SCL local activada y no considerará que ha comenzado el estado de *dato* hasta que efectivamente la línea SCL del bus esté a alta. Por la otra parte, todo esclavo que admita sincronización, estando la línea SCL del bus a baja podrá *pisarla* (poner a baja su SCL local) durante el tiempo que estime oportuno, y sólo liberará la línea SCL del bus en el instante en que esté preparado para gestionar un nuevo bit o carácter. Todo esto puede observarse en la figura 3.9.

Un esclavo receptor podrá utilizar el mecanismo de sincronización en términos bit a bit, carácter a carácter, o cuando y como lo estime oportuno en función de sus intereses.

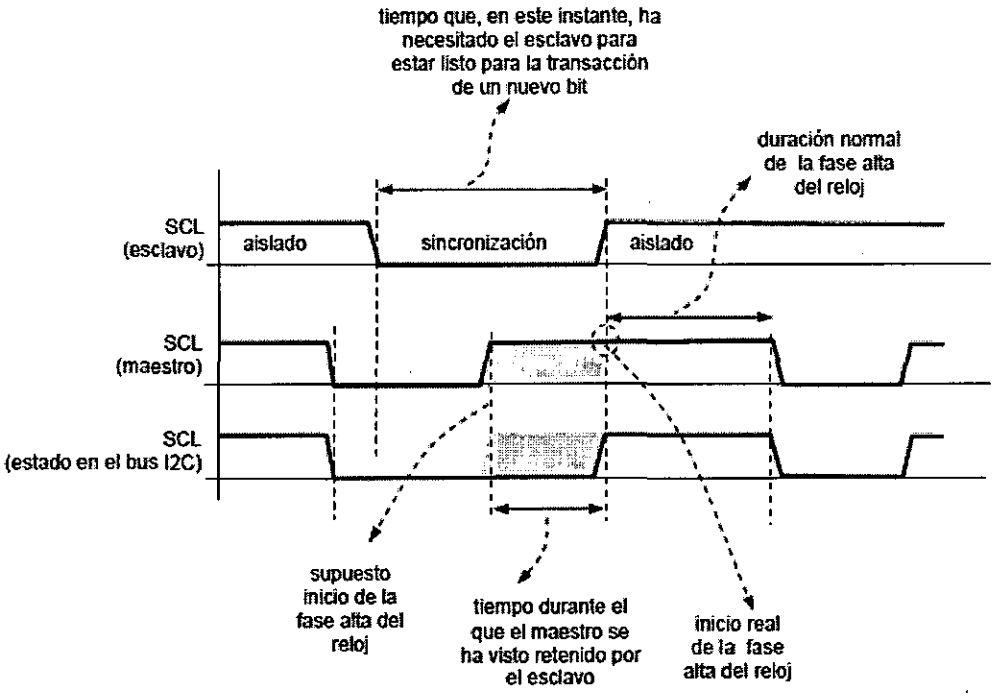


Figura 3. 9 Mecanismo de sincronización del bus I2C

3.3.4. Modo alta velocidad

La norma I2C contempla tres modos de funcionamiento: normal, rápido y de alta velocidad. El primero admite tasas de hasta 100 kilobits por segundo, el segundo –introducido en la revisión 1.0 de 1992– de hasta 400 kbps, y el tercero –introducido en la revisión 2.0 de 1998– de hasta 3'4 megabits por segundo. Existe una compatibilidad descendente, de tal manera que un dispositivo capaz de funcionar en modo de alta velocidad también lo hará en modo rápido, y uno en modo rápido también lo hará en modo normal. Actualmente, todo nuevo dispositivo I2C ha de ser capaz de funcionar al menos en modo rápido; no obstante, los circuitos capaces del modo de alta velocidad resultan una excelente solución para aquellas aplicaciones de altas prestaciones en las que se demande una elevada tasa de transferencia.

Este tipo de circuitos necesita que la electrónica de excitación en los maestros posea unas características especiales con el fin de acelerar las transiciones de baja a alta en la línea de reloj, puesto que de actuar tan sólo la resistencia de elevación sería imposible conseguir las elevadas tasas transferencia de hasta 3'4 megabits. No se olvide que la resistencia de

elevación actuará en las transiciones Low to High como vía de carga de la capacidad concentrada en el bus (hasta 400 pF según la norma), y por ello en la línea SCL se tendrá la típica forma exponencial de carga de un condensador, en el que la constante de tiempo t=RC limitará la máxima velocidad alcanzable.

Los aspectos topológicos así como de tipo eléctrico y temporal se comentarán más adelante, y nos centraremos ahora en comentar los aspectos funcionales y de protocolo que caracterizan al modo de alta velocidad. Por todo lo comentado, los circuitos I2C de alta velocidad denominan a sus señales SDAH y SCLH en lugar de SDA y SCL; no obstante, pueden existir circuitos que ofrezcan ambos tipos de señales, en cuyo caso las normales pueden utilizarse para otros fines si no se utilizan. Esta dualidad de señales resulta útil en topologías con velocidades mixtas.

Por lo que respecta a las características de protocolo, en primer lugar hay que decir que este modo no admite ni el arbitraje ni la sincronización bit a bit puesto que tal cuestión iría en contra del objetivo perseguido de conseguir la máxima velocidad posible de transferencia. Esto es, todo dispositivo maestro capaz de la alta velocidad deberá realizar transacciones sólo con circuitos también capaces de la alta velocidad. Por supuesto, si un maestro es capaz de la alta velocidad pero funciona en un modo inferior (rápido o normal) entonces podrá dirigirse a cualquier circuito capaz de tales modos inferiores. El arbitraje se realiza en un modo que no es de alta velocidad, como se analizará más adelante.

Para evitar cargar excesivamente el bus (a mayor número de circuitos mayor capacidad equivalente existirá en él) los circuitos normales y rápidos se pueden separar de los de alta velocidad mediante sendos *puentes* en SDA y SCL, que no son más que unos circuitos especiales diseñados para tal caso.

3.4. Protocolo de comunicación Bluetooth

En 1994, la empresa L. M. Ericsson se interesó en conectar sus teléfonos móviles y otros dispositivos; por ejemplo PDAs (Personal Digital Assistant), sin necesidad de cables. En conjunto con otras cuatro empresas: IBM, Intel, Nokia y Toshiba, formó un **SIG** (grupo de interés especial, es decir, un consorcio) con el propósito de desarrollar un estándar inalámbrico para interconectar computadoras, dispositivos de comunicaciones y accesorios a través de radios inalámbricos de bajo consumo de energía, corto alcance y económicos. Al proyecto se le asignó el nombre ***Bluetooth***, en honor de Harald Blaatand (Bluetooth) II (940-981), un rey vikingo que unificó Dinamarca y Noruega, también sin necesidad de cables.

Aunque la idea original eran tan sólo prescindir de cables entre dispositivos, su alcance se expandió rápidamente al área de las **LANs** inalámbricas. Aunque esta expansión le dio más utilidad al estándar, también provocó el surgimiento de competencia con el 802.11. Para empeorar las cosas, los dos sistemas interfieren entre sí en el ámbito eléctrico.

Asimismo, vale la pena hacer notar que Hewlett-Packard introdujo hace algunos años una red infrarroja para conectar periféricos de computadora sin cables, pero en realidad nunca alcanzó popularidad. Sin desanimarse por esto, el SIG de Bluetooth emitió en julio de 1999 una especificación de 1500 páginas acerca de V1.0. Un poco después, el grupo de estándares del IEEE que se encarga de las redes de área personal inalámbricas, 802.15, adoptó como base el documento sobre Bluetooth y empezó a trabajar en él. A pesar de que podría parecer extraño estandarizar algo que ya cuenta con una especificación bien detallada, sin implementaciones incompatibles que tengan que armonizarse, la historia demuestra que al existir un estándar abierto manejado por un cuerpo neutral como el IEEE con frecuencia se estimula el uso de una tecnología. Para ser un poco más precisos, debe apuntarse que la especificación de Bluetooth está dirigida a un sistema completo, desde la capa física a la capa de aplicación. El comité 802.15 del IEEE estandariza solamente las capas física y la de enlace de datos; el resto de la pila de protocolos está fuera de sus estatutos.

3.4.1. Arquitectura de Bluetooth

Empecemos nuestro análisis del sistema Bluetooth con un rápido vistazo de sus elementos y de su propósito. La unidad básica de un sistema Bluetooth es una **piconet,** que consta de un nodo maestro y hasta siete nodos esclavos activos a una distancia de 10 metros. En una misma sala (grande) pueden encontrarse varias ***piconets*** y se pueden conectar mediante un nodo puente, como se muestra en la figura 3.10. Un conjunto de ***piconets*** interconectadas se denomina **scatternet.**

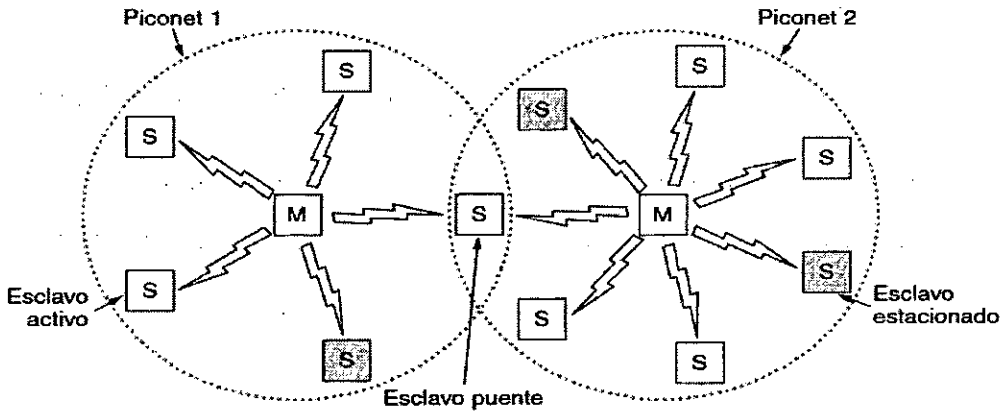


Figura 3. 10 Formación de una scatternet. Dos piconet se pueden enlazar para formar una scatternet

Además de los siete nodos esclavos activos de una ***piconet,*** puede haber hasta 255 nodos estacionados en la red. Éstos son dispositivos que el nodo maestro ha cambiado a un estado de bajo consumo de energía o también llamado ***"modo espera"***, para reducir el desgaste innecesario de sus pilas. Lo único que un dispositivo en estado estacionado puede hacer es responder a una señal de activación por parte del maestro. Existen otros dos modos los cuales puede adoptar los dispositivos esclavos como lo son el ***"modo sniff"***, donde se reduce el ciclo de trabajo de una estación esclava, ya que sólo escucha cada M slots, siendo el valor de M negociado con el gestor de enlace, y el ***"modo hold"***, el cual es útil cuando un dispositivo no necesita participar activamente en el canal, pero sí quiere permanecer sincronizado.

La razón para el diseño maestro/esclavo es que los diseñadores pretendían facilitar la implementación de chips Bluetooth completos por

debajo de 5 dólares. La consecuencia de esta decisión es que los esclavos son sumamente pasivos y realizan todo lo que los maestros les indican. En esencia, una *piconet* es un sistema TDM (Multiplexación por División de Tiempo) centralizado, en el cual el maestro controla el reloj y determina qué dispositivo se comunica en un momento determinado. Todas las comunicaciones se realizan entre el maestro y el esclavo; no existe comunicación directa de esclavo a esclavo.

3.4.2. Aplicaciones de Bluetooth

La mayoría de los protocolos de red sólo proporcionan canales entre las entidades que se comunican y permiten a los diseñadores de aplicaciones averiguar para qué desean utilizarlos. Por ejemplo, el 802.11 no especifica si los usuarios deben utilizar sus computadoras portátiles para leer correo electrónico, navegar por la Red o cualquier otro uso. En contraste, la especificación Bluetooth V1.1 designa el soporte de 13 aplicaciones en particular y proporciona diferentes pilas de protocolos para cada una. Desgraciadamente, este enfoque conlleva una gran complejidad, que aquí omitiremos.

Nombre	Descripción
Acceso genérico	Procedimientos para el manejo de enlaces
Descubrimiento de servicios	Protocolo para descubrir los servicios que se ofrecen
Puerto serie	Reemplazo para un cable de puerto serie
Intercambio genérico de objetos	Define la relación cliente-servidor para el traslado de objetos
Acceso a LAN	Protocolo entre una computadora móvil y una LAN fija
Acceso telefónico a redes	Permite que una computadora portátil realice una llamada por medio de un teléfono móvil
Fax	Permite que un fax móvil se comunique con un teléfono móvil
Telefonía inalámbrica	Conecta un handset (teléfono) con su estación base local
Intercom (Intercomunicador)	Walkie-talkie digital
Headset (Diadema telefónica)	Posibilita la comunicación de voz sin utilizar las manos
Envío de objetos	Ofrece una manera de intercambiar objetos simples
Transferencia de archivos	Proporciona una característica para transferencia de archivos más general
Sincronización	Permite a un PDA sincronizarse con otra computadora

Figura 3. 11 Perfiles de Bluetooth

- El ***perfil de acceso genérico,*** no es realmente una aplicación, sino más bien la base sobre la cual se construyen las aplicaciones; su tarea principal es ofrecer una manera para establecer y mantener enlaces (canales) seguros entre el maestro y los esclavos.
- El ***perfil de descubrimiento de servicios*** también es relativamente genérico; los dispositivos lo utilizan para descubrir qué servicios ofrecen otros dispositivos. Se espera que todos los dispositivos Bluetooth implementen estos dos perfiles. Los restantes son opcionales.
- El ***perfil de puerto serie*** es un protocolo de transporte que la mayoría de los perfiles restantes utiliza. Emula una línea serie y es especialmente útil para aplicaciones heredadas que requieren una línea serie.
- El ***perfil de intercambio genérico*** define una relación cliente-servidor para el traslado de datos. Los clientes inician operaciones, pero tanto un cliente como un servidor pueden fungir como esclavo. Al igual que el perfil de puerto serie, es la base para otros perfiles.
- El ***perfil de acceso a LAN*** permite a un dispositivo Bluetooth conectarse a una red fija; este perfil es competencia directa del estándar 802.11.
- El ***perfil de acceso telefónico a redes*** fue el propósito original de todo el proyecto; permite a una computadora portátil conectarse a un teléfono móvil que contenga un módem integrado, sin necesidad de cables.
- El ***perfil de fax*** es parecido al de acceso telefónico a redes, excepto que posibilita a máquinas de fax inalámbricas enviar y recibir faxes a través de teléfonos móviles sin que exista una conexión por cable entre ambos.
- El ***perfil de telefonía inalámbrica*** proporciona una manera de conectar el *handset* (teléfono) de un teléfono inalámbrico a la estación base. En la actualidad, la mayoría de los teléfonos inalámbricos no se puede utilizar también como teléfonos móviles, pero quizá en el futuro se puedan combinar los teléfonos inalámbricos y los móviles.

- El ***perfil intercom*** hace posible que dos teléfonos se conecten como *walkie-talkies*.
- ***El perfil headset*** (diadema telefónica) se puede realizar comunicación de voz entre la diadema telefónica y su estación base, por ejemplo, para comunicarse telefónicamente sin necesidad de utilizar las manos al manejar un automóvil.

Los tres perfiles restantes sirven para intercambiar objetos entre dos dispositivos inalámbricos, como tarjetas de presentación, imágenes o archivos de datos. En particular, el propósito del perfil de sincronización es cargar datos en un PDA o en una computadora portátil cuando se está fuera de casa y de recabar estos datos al llegar a casa.

¿Era realmente necesario explicar en detalle todas estas aplicaciones y proporcionar diferentes pilas de protocolos para cada una? Tal vez no, pero esto se debió a que fueron diversos grupos de trabajo los que diseñaron las diferentes partes del estándar y cada uno se enfocó en su problema específico y generó su propio perfil. Quizá dos pilas de protocolos habrían sido suficientes en lugar de 13, una para la transferencia de archivos y otra para posibilitar el flujo continuo de la comunicación en tiempo real.

3.4.3. La pila de protocolos de Bluetooth

El estándar Bluetooth cuenta con muchos protocolos agrupados con poco orden en capas. La estructura de capas no sigue el modelo OSI, el modelo TCP/IP, el modelo 802 o algún otro modelo conocido. Sin embargo, el IEEE se encuentra modificando actualmente Bluetooth para ajustarlo al modelo 802. En la figura 3.12 se muestra la arquitectura básica de protocolos de Bluetooth tal como la modificó el comité 802.

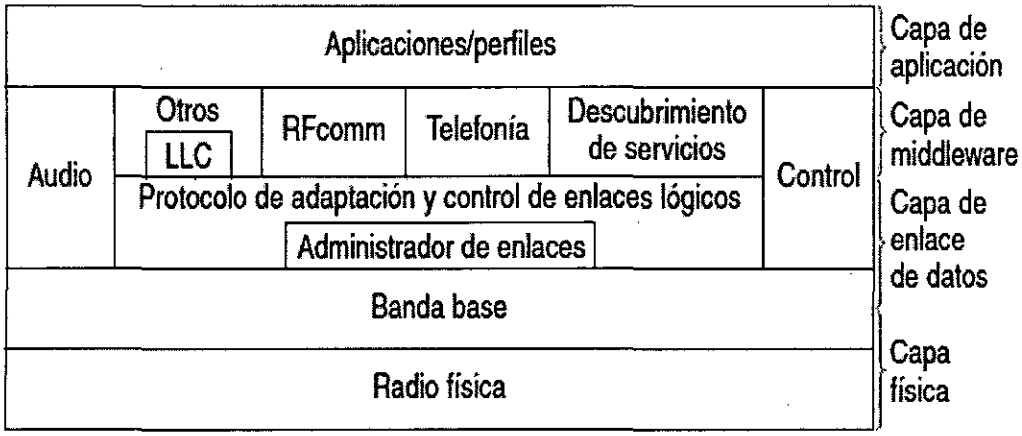


Figura 3. 12 Versión 802.15 de la arquitectura de protocolos Bluetooth

La capa inferior es la capa de radio física, la cual es bastante similar a la capa física de los modelos OSI y 802. Se ocupa de la transmisión y la modulación de radio. Aquí, gran parte del interés se enfoca en el objetivo de lograr que el sistema tenga un costo bajo para que pueda entrar al mercado masivo. La capa de banda base tiene algunos puntos en común con la subcapa MAC, aunque también incluye elementos de la capa física. Se encarga de la manera en que el maestro controla las ranuras de tiempo y de que éstas se agrupen en tramas. A continuación se encuentra una capa con un grupo de protocolos un tanto relacionados.

El administrador de enlaces se encarga de establecer canales lógicos entre dispositivos, incluyendo administración de energía, autenticación y calidad de servicio. El protocolo de adaptación y control de enlaces lógicos (también conocido como L2CAP) aísla a las capas superiores de los detalles de la transmisión. Es análogo a la subcapa LLC del estándar 802, pero difiere de ésta en el aspecto técnico. Como su nombre indica, los protocolos de audio y control se encargan del audio y el control, respectivamente. Las aplicaciones pueden acceder a ellos de manera directa sin necesidad de pasar por el protocolo L2CAP.

La siguiente capa hacia arriba es la de middleware, que contiene una mezcla de diferentes protocolos. El IEEE incorporó aquí la subcapa LLC del 802 para ofrecer compatibilidad con las redes 802. Los protocolos ***RFcomm***, de telefonía y de descubrimiento de servicios son nativos. ***RFcomm (comunicación de Radio Frecuencia)*** es el protocolo que emula el puerto serie estándar de las PCs para la conexión de teclados, ratones y módems, entre otros dispositivos. Su propósito es permitir que dispositivos heredados lo utilicen con facilidad. El protocolo de telefonía es de tiempo real y está destinado a los tres perfiles orientados a voz. También se encarga del establecimiento y terminación de llamadas. Por último, el protocolo de descubrimiento de servicios se emplea para localizar servicios dentro de la red.

En la capa superior es donde se ubican las aplicaciones y los perfiles, que utilizan a los protocolos de las capas inferiores para realizar su trabajo. Cada aplicación tiene su propio subconjunto dedicado de protocolos. Por lo general, los dispositivos específicos, como las diademas telefónicas, contienen únicamente los protocolos necesarios para su aplicación.

3.4.4. La capa de radio de Bluetooth

La capa de radio traslada los bits del maestro al esclavo, o viceversa. Es un sistema de baja potencia con un rango de 10 metros que opera en la banda ISM de 2.4 GHz. La banda se divide en 79 canales de 1 MHz cada uno. La modulación es por desplazamiento de frecuencia, con 1 bit por Hz, lo cual da una tasa de datos aproximada de 1 Mbps, pero gran parte de este espectro la consume la sobrecarga.

Para asignar los canales de manera equitativa, el espectro de saltos de frecuencia se utiliza a 1600 saltos por segundo y un tiempo de permanencia de 625 μseg. Todos los nodos de una *piconet* saltan de manera simultánea, y el maestro establece la secuencia de salto. Debido a que tanto el 802.11 como Bluetooth operan en la banda ISM de 2.4 GHz en los mismos 79 canales, interfieren entre sí. Puesto que Bluetooth salta mucho más rápido que el 802.11, ***es más probable que un dispositivo Bluetooth dañe las***

transmisiones del 802.11 que en el caso contrario. Como el 802.11 y el 802.15 son estándares del IEEE, éste busca una solución para el problema, aunque no es tan sencilla porque ambos sistemas utilizan la banda ISM por la misma razón: no se requiere licencia para su uso.

El estándar 802.11a emplea la otra banda ISM (5 GHz), pero tal estándar tiene un rango mucho más corto que el 802.11b (debido a la física de las ondas de radio), por lo cual el 802.11a no es una solución idónea en todos los casos. Algunas empresas han recurrido a la prohibición total de Bluetooth para solucionar el problema. Una solución de mercado es que la red más potente (en los aspectos político y económico, no eléctrico) solicite que la parte más débil modifique su estándar para dejar de interferir con ella.

3.4.5. La capa de banda base de Bluetooth

La capa de banda base de Bluetooth es lo más parecido a una subcapa MAC. Esta capa convierte el flujo de bits puros en tramas y define algunos formatos clave. En la forma más sencilla, el maestro de cada *piconet* define una serie de ranuras de tiempo de 625 μseg y las transmisiones del maestro empiezan en las ranuras pares, y las de los esclavos, en las ranuras impares.

Ésta es la tradicional multiplexación por división de tiempo, en la cual el maestro acapara la mitad de las ranuras y los esclavos comparten la otra mitad. Las tramas pueden tener 1, 3 o 5 ranuras de longitud. La sincronización de saltos de frecuencia permite un tiempo de asentamiento de 250-260 μseg por salto para que los circuitos de radio se estabilicen. Es posible un asentamiento más rápido, pero a un mayor costo.

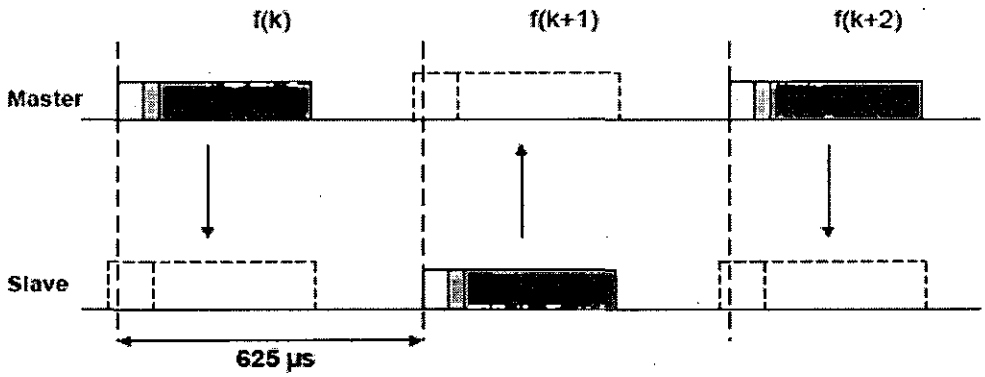


Figura 3. 13 Multiplexacion por división de tiempo y trasmisión Bluetooth

Para una trama de una sola ranura, después del asentamiento, se desechan 366 de los 625 bits. De éstos, 126 se utilizan para un código de acceso y el encabezado, y 240 para los datos. Cuando se enlazan cinco ranuras, sólo se necesita un periodo de asentamiento y se utiliza uno ligeramente más corto, de tal manera que de los $5 \times 625 = 3125$ bits de las cinco ranuras de tiempo, 2781 se encuentran disponibles para la capa de banda base. Así, las tramas más grandes son mucho más eficientes que las de una sola ranura.

Cada trama se transmite por un canal lógico, llamado **enlace,** entre el maestro y un esclavo. Hay dos tipos de enlaces:

- **ACL (Asíncrono no Orientado a la Conexión):** se utiliza para datos conmutados en paquetes disponibles a intervalos irregulares. Estos datos provienen de la capa L2CAP en el nodo emisor y se entregan en la capa L2CAP en el nodo receptor. El tráfico ACL se entrega sobre la base de mejor esfuerzo. No hay garantías. Las tramas se pueden perder y tienen que retransmitirse. Un esclavo puede tener sólo un enlace ACL con su maestro.
- **SCO (Síncrono Orientado a la Conexión):** utilizada para datos en tiempo real, como ocurre en las conexiones telefónicas. A este tipo de canal se le asigna una ranura fija en cada dirección. Por la importancia del tiempo en los enlaces SCO, las tramas que se envían a través de ellos nunca se retransmiten. En vez de ello, se puede utilizar la corrección de errores hacia delante (o corrección de errores sin canal de retorno) para conferir una confiabilidad alta. Un esclavo puede establecer hasta tres enlaces SCO con su maestro. Cada enlace de este tipo puede transmitir un canal de audio PCM de 64,000 bps.

3.4.6. La capa L2CAP de Bluetooth

La capa L2CAP tiene tres funciones principales. Primera, acepta paquetes de hasta 64 KB provenientes de las capas superiores y los divide en tramas para transmitirlos. Las tramas se re ensamblan nuevamente en paquetes en el otro extremo.

Segunda, maneja la multiplexión y desmultiplexión de múltiples fuentes de paquetes. Cuando se re ensambla un paquete, la capa L2CAP determina cuál protocolo de las capas superiores lo manejará, por ejemplo, el RFcomm o el de telefonía.

Tercera, la capa L2CAP se encarga de la calidad de los requerimientos de servicio, tanto al establecer los enlaces como durante la operación normal. Asimismo, durante el establecimiento de los enlaces se negocia el tamaño máximo de carga útil permitido, para evitar que un dispositivo que envíe paquetes grandes sature a uno que reciba paquetes pequeños. Esta característica es importante porque no todos los dispositivos pueden manejar paquetes de 64 KB.

3.4.7. Estructura de la trama de Bluetooth

Existen diversos formatos de trama, el más importante de los cuales se muestra en la figura 26. Empieza con un código de acceso que identifica al maestro, cuyo propósito es que los esclavos que se encuentren en el rango de alcance de dos maestros sepan cuál tráfico es para ellos.

A continuación se encuentra un encabezado de 54 bits que contiene campos comunes de la subcapa MAC. Luego está el campo de datos, de hasta 2744 bits (para una transmisión de cinco ranuras). Para una sola ranura de tiempo, el formato es el mismo excepto que el campo de datos es de 240 bits.

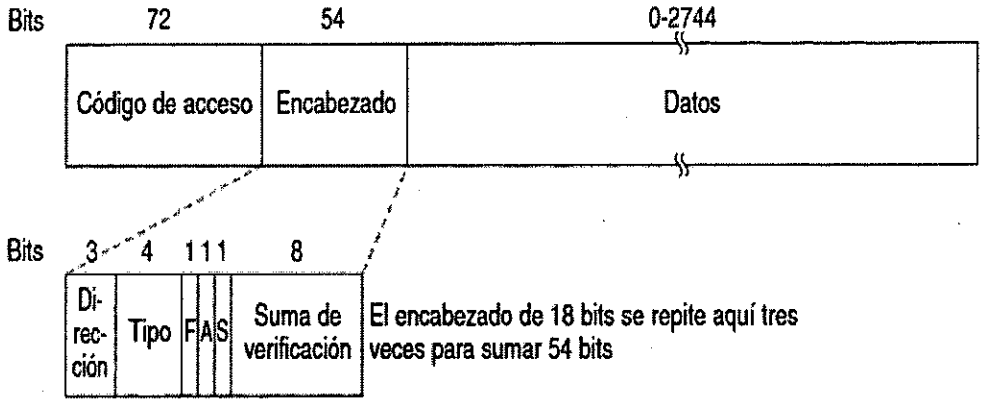


Figura 3. 14 Formato de trama de datos típica de Bluetooth

Demos un rápido vistazo al encabezado. El campo *Dirección* identifica a cuál de los ocho dispositivos activos está destinada la trama. El campo *Tipo* indica el tipo de trama (ACL, SCO, de sondeo o nula), el tipo de corrección de errores que se utiliza en el campo de datos y cuántas ranuras de longitud tiene la trama. Un esclavo establece el bit *F* (de flujo) cuando su búfer está lleno y no puede recibir más datos. Ésta es una forma primitiva de control de flujo. El bit *A* (de confirmación de recepción) se utiliza para incorporar un ACK en una trama. El bit *S* (de secuencia) sirve para numerar las tramas con el propósito de detectar retransmisiones.

El protocolo es de parada y espera, por lo que 1 bit es suficiente. A continuación se encuentra el encabezado *Suma de verificación* de 8 bits. Todo el encabezado de 18 bits se repite tres veces para formar el encabezado de 54 bits que se aprecia en la figura 26. En el receptor, un circuito sencillo examina las tres copias de cada bit. Si son las mismas, el bit es aceptado. De lo contrario, se impone la opinión de la mayoría. De esta forma, 54 bits de capacidad de transmisión se utilizan para enviar 10 bits de encabezado. Esto se debe a que es necesaria una gran cantidad de redundancia para enviar datos de manera confiable en un entorno con ruido mediante dispositivos de bajo costo y baja potencia (2.5 mW) con poca capacidad de cómputo.

En el campo de datos de las tramas ACL se utilizan varios formatos. Sin embargo, las tramas SCO son más sencillas: el campo de datos siempre es de 240 bits. Se definen tres variantes, que permiten 80, 160 y 240 bits de carga útil real, y el resto se utiliza para corrección de errores. En la versión más confiable (carga útil de 80 bits), el contenido se repite tres veces, al igual que el encabezado. Debido a que el esclavo podría utilizar solamente las ranuras impares, obtiene 800 ranuras por segundo, de la misma manera que el maestro. Con una carga útil de 80 bits, la capacidad de canal del esclavo es de 64,000 bps y la del maestro también es de 64,000 bps, exactamente la necesaria para un canal de voz PCM dúplex total (razón por la cual se eligió una tasa de saltos de 1600 saltos por segundo).

Estas cifras indican que un canal de voz dúplex total con 64,000 bps en cada dirección y el formato más confiable satura totalmente la *piconet* a

pesar de un ancho de banda puro de 1 Mbps. En la variante menos confiable (240 bits por ranura sin redundancia a este nivel), se pueden soportar al mismo tiempo tres canales de voz dúplex total, debido a lo cual se permite un máximo de tres enlaces SCO por esclavo.

3.5. Bluetooth 4.0 LE

A mitades del 2010 el Bluetooth SIG anuncio la adopción formal de la especificación para la versión Bluetooth 4.0 LE. Entre las características que distinguen esta especificación están:

- Tres modos de consumo de energía: Pico ultra-bajo, promedio e inactivo.
- Habilidad de correr por años en una batería de reloj estándar.
- Bajo costo.
- Interoperabilidad de múltiples vendedores.
- Rango mejorado. (Bluetooth-SIG, 2013)

Con la incursión de esta versión se establecen dos formas para el sistema de la tecnología Bluetooth, las cuales son Basic Rate (BR) y Low Energy (LE). En el Basic Rate se tienen como opciones el EDR, un MAC alternativo y diferentes extensiones para la capa física (PHY). De esta forma en BR se ofrecen conexiones síncronas y asíncronas donde la tasa de transferencia de datos para Basic Rate es de 721.2 kb/s, 2.1 Mb/s con la opción EDR y hasta 24 Mb/s cuando se tiene el modo HS utilizando AMPs con 802.11. Mientras tanto el sistema LE se agrega características para productos que requieren un menor consumo de corriente y menor costo que BR/EDR. También, se tomó en cuenta para el diseño de LE los casos y aplicaciones que usan una menor tasa de transferencia de datos y tiene un menor ciclo de trabajo.

Los dispositivos que tengan ambos sistemas pueden comunicarse con otros dispositivos que utilicen ambos sistemas o que solo utilicen uno de ellos. El sistema principal de Bluetooth consiste en un Host y uno o más controladores, y en esta versión se definen dos tipos de controladores los cuales con los controladores primarios y los controladores secundarios. De esta manera solo es necesario un controlador primario cuando se tienen los modos BR/EDR, LE o la combinación de estos dos anteriores en un solo controlador. Mientras que los controladores secundarios se utilizan cuando se utiliza uno o más AMPs.

3.5.1. Funcionamiento

El radio de la tecnología LE trabaja en la banda ISM de 2.4 GHz al igual que en BR. En este sistema se emplea un transceptor con saltos en frecuencia, de esta manera se combate la interferencia y se emplean muchas portadoras FHSS. En esta tecnología se soporta una tasa de transferencia de datos de hasta 1 Mb/s.

En Bluetooth LE se emplean 2 tipos de esquemas para acceso múltiple: FDMA (acceso múltiple por división de frecuencia) que utiliza 40 canales físicos separados por 2 MHz, en donde 3 son utilizados para realizar anuncios y 37 son para datos. El otro es TDMA (acceso múltiple por división de tiempo) en donde su esquema se basa en que un dispositivo transmite un paquete en un momento predeterminado y otro dispositivo correspondiente responde con un paquete después de un intervalo predeterminado. En esta tecnología se subdivide el canal físico en unidades de tiempo conocidas como eventos.

De esta manera los dispositivos se transmiten los datos por medio de paquetes que son posicionados en estos eventos de tiempo. Existen 2 tipos de eventos: los eventos de anuncios y los eventos de conexión.

3.5.1.1. Formato del paquete

El formato de los paquetes que se envían en la tecnología Bluetooth LE es distinto al utilizado en BR/EDR. En la figura 3.5 se puede observar el formato de un paquete de datos de la tecnología Bluetooth LE. En la Dirección de Acceso (Access Address) se contiene el código de acceso del enlace físico. El Encabezado PDU (PDU Header) contiene el transporte lógico y los identificadores del enlace lógico. La Carga útil PDU (PDU Payload) contiene las señales y tramas de la capa lógica, además de otros datos del usuario.

Figura 3. 15 Formato del paquete Bluetooth 4.0

3.6. Protocolo de comunicación USB

El Universal Serial Bus (bus universal en serie) o Conductor Universal en Serie, abreviado comúnmente USB, es un puerto que sirve para conectar periféricos a una computadora. Fue creado en 1996 por siete empresas: IBM, Intel, Northern Telecom, Compaq, Microsoft, Digital Equipment Corporation y NEC. El estándar incluye la transmisión de energía eléctrica al dispositivo conectado.

El diseño del USB tenía en mente eliminar la necesidad de adquirir tarjetas separadas para poner en los puertos bus ISA o PCI, y mejorar las capacidades ***plug-and-play*** permitiendo a esos dispositivos ser conectados o desconectados al sistema sin necesidad de reiniciar.

Cuando se conecta un nuevo dispositivo, el servidor lo enumera y agrega el software necesario para que pueda funcionar.

El USB puede conectar los periféricos como ratones, teclados, escáneres, cámaras digitales, teléfonos móviles, reproductores multimedia, impresoras, discos duros externos, tarjetas de sonido, sistemas de adquisición de datos y componentes de red. Para dispositivos multimedia como escáneres y cámaras digitales, el USB se ha convertido en el método estándar de conexión.

3.6.1. Clasificación

Los dispositivos USB se clasifican en cuatro tipos según su velocidad de transferencia de datos:

- **Baja velocidad (1.0):** Tasa de transferencia de hasta 1'5 Mbps (192 KB/s). Utilizado en su mayor parte por dispositivos HID (Human Interface Device) como los teclados, los ratones y los joysticks.
- **Velocidad completa (1.1):** Tasa de transferencia de hasta 12 Mbps (1'5 MB/s). Ésta fue la más rápida antes de la especificación USB 2.0, y muchos dispositivos fabricados en la actualidad trabajan a esta velocidad. Estos dispositivos dividen el ancho de banda de la conexión USB entre ellos, basados en un algoritmo de búferes FIFO.

- **Alta velocidad (2.0):** Tasa de transferencia de hasta 480 Mbps (60 MB/s) pero por lo general de hasta 125Mbps (16MB/s). Está presente casi en el 99% de los ordenadores actuales. El cable USB 2.0 dispone de cuatro líneas, un par para datos, una de corriente y una de toma de tierra.
- **Súper velocidad (3.0):** Cuenta con tasa de transferencia de hasta 4.8 Gbps (600 MB/s). Esta especificación fue lanzada a mediados de 2009 por Intel, de acuerdo con información recabada de Internet. A la vez, la intensidad de la corriente trepa de los 500 a los 900 miliamperios, que sirve para abastecer a un teléfono móvil o un reproductor audiovisual portátil en menos tiempo.
  Por otro lado, aumenta la velocidad en la transmisión de datos, ya que en lugar de funcionar con tres líneas, lo hace con cinco. De esta manera, dos líneas se utilizan para enviar, otras dos para recibir, y una quinta se encarga de suministrar la corriente; así, el tráfico es bidireccional (Full dúplex). La versión 3.0 de este conector universal es 10 veces más rápida que la anterior. Aquellos que tengan un teclado o un ratón de la versión anterior no tendrán problemas de compatibilidad, ya que el sistema lo va a reconocer al instante, aunque no podrán beneficiarse de los nuevos adelantos de este puerto USB serial bus.

3.6.2. Funcionamiento

Es un bus basado en el paso de un testigo, semejante a otros buses como los de las redes locales en anillo con paso de testigo y las redes FDDI. El controlador USB distribuye testigos por el bus. El dispositivo cuya dirección coincide con la que porta el testigo responde aceptando o enviando datos al controlador. Este también gestiona la distribución de energía a los periféricos que lo requieran. Utiliza 4 líneas mayadas, de las cuales 2 son para alimentación; VCC y GND respectivamente, y las otras dos, forman un par trenzado con una impedancia característica de 90Ω, donde las señales se transmiten de forma diferencial, eliminando así los ruidos e interferencias.

Los puertos USB admiten dispositivos Plug and Play de conexión en caliente. Por lo tanto, los dispositivos pueden conectarse sin apagar el equipo (conexión en caliente). Cuando un dispositivo está conectado al host, detecta cuando se está agregando un nuevo elemento gracias a un cambio de tensión entre los hilos D+ y D-. En ese momento, el equipo envía una señal de inicialización al dispositivo durante 10 ms para después suministrarle la corriente eléctrica mediante los hilos GND y VCC. A continuación, se le suministra corriente eléctrica al dispositivo y temporalmente se apodera de la dirección predeterminada (dirección 0). La siguiente etapa consiste en brindarle la dirección definitiva (éste es el procedimiento de lista). Para hacerlo, el equipo interroga a los dispositivos ya conectados para poder conocer sus direcciones y asigna una nueva, que lo identifica por retorno. Una vez que cuenta con todos los requisitos necesarios, el host puede cargar el driver adecuado.

3.6.3. Topología

El Universal Serial Bus conecta los dispositivos USB con el host USB. La interconexión física USB es una topología de estrellas apiladas donde un hub es el centro de cada estrella. Cada segmento de cable es una conexión punto-a-punto entre el host y los hubs o función, o un hub conectado a otro hub o función. En la raíz o vértice de las capas, está el controlador anfitrión o host que controla todo el tráfico que circula por el bus. Esta topología permite a muchos dispositivos conectarse a un único bus lógico sin que los dispositivos que se encuentran más abajo en la pirámide sufran retardo.

A diferencia de otras arquitecturas, USB no es un bus de almacenamiento y envío, de forma que no se produce retardo en el envío de un paquete de datos hacia capas inferiores. El número máximo de dispositivos que puede conectar por USB es de 127, pero debido a las constantes de tiempo permitidas para los tiempos de propagación del hub y el cable, el número máximo de capas permitido es de siete (incluida la capa raíz) con un máximo de longitud de cable entre el hub y el dispositivo de 5 metros. Cabe destacar que en siete capas, sólo se soportan cinco hubs que no sean raíz en una ruta de comunicación entre el host y cualquier dispositivo. Un dispositivo

compuesto ocupa dos capas, por eso, no puede ser activado si está acoplado en la última capa de nivel siete. Sólo las funciones pueden ponerse en este nivel.

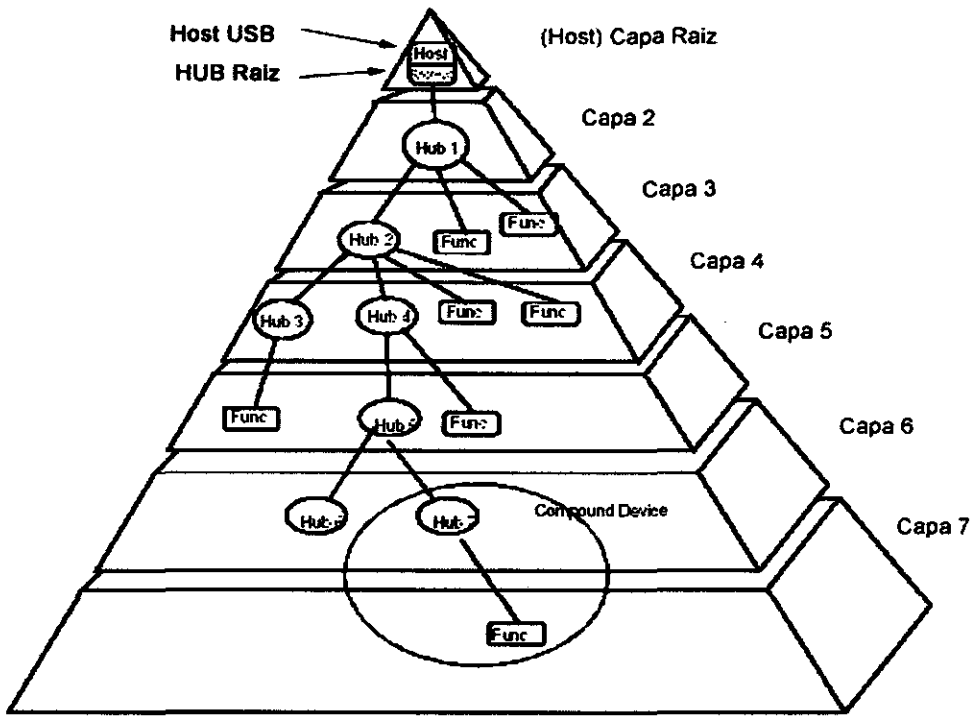


Figura 3. 16 Capas del Bus USB

La topología del bus USB se puede dividir en 3 partes:

- Capa física
- Capa lógica
- Relación cliente-servidor

3.6.3.1. Capa Física

La arquitectura física del USB se centra en las piezas de plástico y de metal con las que el usuario debe tratar para construir un entorno USB. Los dispositivos están conectados físicamente al host a través de una topología en estrella, como se ilustra en la figura 23. Los puntos de acople están provistos de una clase de dispositivos USB

llamados hubs, los cuales tienen puntos de acople adicionales llamados puertos.

Estos hubs se conectan a otros dispositivos a través de enlaces (cables de cuatro hilos). El host proporciona uno o más puntos de acople a través del hub raíz. Para prevenir los acoples circulares, se impone una estructura ordenada por capas en la topología de estrella y como resultado se obtiene una configuración al estilo de un árbol como se ve en la figura.

Todas las comunicaciones físicas son iniciadas por el host. Esto quiere decir que cada milisegundo, o en cada ventana de tiempo que el bus lo permita, el host preguntará por nuevos dispositivos en el bus USB. Además el host inicia todas las transacciones físicas y soporta todas las transferencias de datos sobre la capa física

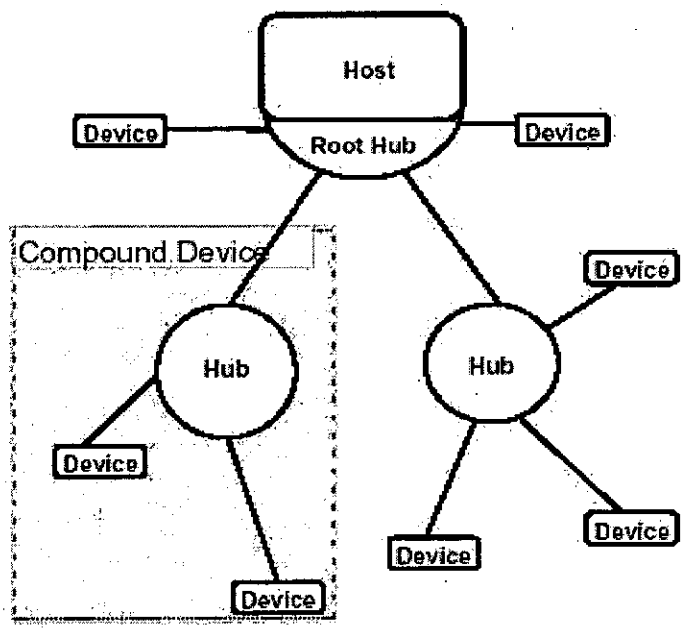


Figura 3. 17 Topología física del bus USB

Cada entorno físico USB está compuesto por cinco tipos de componentes:

- **El host:** El host es el sistema de computación completo, incluyendo el software y el hardware, sobre el cual se sostiene el USB.

  El host tiene la habilidad de procesar y gestionar los cambios de configuración que puedan ocurrir en el bus durante su funcionamiento. El host gestiona el sistema y los recursos del bus como el uso de la memoria del sistema, la asignación del ancho de banda del bus y la alimentación del bus. El host también ayuda al usuario con la configuración automática de los dispositivos conectados y reaccionando cuando son desconectados.

  Un host puede soportar uno o más buses USB. El host gestiona cada bus independientemente de los demás. Los recursos específicos del bus como el ancho de banda asignado son únicos a cada bus. Cada bus está conectado al host a través de un controlador del host.

  Sólo hay un host en cualquier sistema USB. Desde el interfaz USB hasta el sistema de host del ordenador es lo que se le llama controlador de host y puede estar implementado como combinación de hardware, ***firmware*** o software. Integrado dentro del sistema de host hay un ***Hub raíz*** que provee de un mayor número de puntos de acople al sistema.

  Siempre que es posible, el software del USB usa el interfaz existente del sistema de host para gestionar las interacciones superiores. Por ejemplo, si un sistema de host usa la Gestión de Energía Avanzada (APM), el software del USB conecta al APM para interceptar, suspender las notificaciones.

- **El controlador del host:** El controlador de host está formado por el hardware y el software que permite a los dispositivos USB ser conectados al host. Este controlador es el agente iniciador del bus, es decir es el que comienza las transferencias en el bus. El controlador de bus es el maestro en un bus USB. Otros buses como PCI, permiten la presencia de múltiples maestros donde cada uno arbitra sus accesos al bus. El la arquitectura USB sólo hay un controlador de host por cada bus USB y por eso no hay arbitración para el acceso al bus.

Como las transferencias de datos de los dispositivos pueden ser basadas en datos o en la disponibilidad espacial del dispositivo, la mayoría de los controladores de host están implementados como dispositivos maestros de bus PCI. Esto permite al controlador de host iniciar una transferencia de datos en el bus del sistema cuando le sea necesario, sin requerir la intervención del host de la CPU para cada transferencia. El controlador se comporta como un bus maestro PCI multicanal programable para dar soporte a las necesidades de transferencia de datos de múltiples dispositivos conectados al bus USB.

La figura 3.18 muestra una vista conceptual del controlador de bus USB.

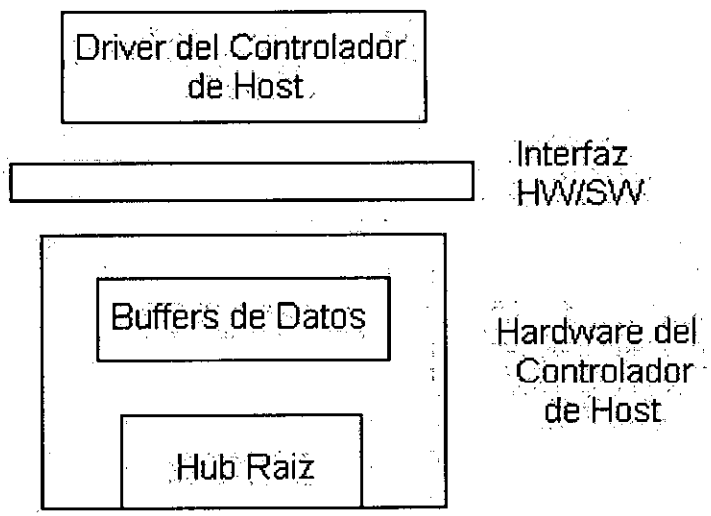


Figura 3. 18 Controlador del bus USB

La parte software consiste en el driver del controlador de host (HCD). Este software interactúa con el hardware del controlador de host a través del interfaz hardware/software.

La parte hardware del controlador de host consiste en un hub raíz que proporciona los puertos USB y los buffers de datos (colas) donde son almacenadas cuando son movidas a/desde memoria.

El host USB interactúa con los dispositivos USB a través del controlador. Las funciones básicas del controlador de host son:

- Detectar la inserción o desconexión de dispositivos USB
- Gestionar el flujo de control entre el host y los dispositivos
- Gestionar el flujo de datos entre el host y los dispositivos
- Coleccionar estadísticas de actividad y estado
- Proveer una cantidad limitada de energía a los dispositivos

- **Dispositivos USB:** Un dispositivo es una colección de funcionalidad que lleva a cabo algún propósito de utilidad. Por ejemplo, un dispositivo podría ser un ratón, un teclado, una cámara, etc. Pueden haber múltiples dispositivos simultáneamente en el mismo bus. Cada dispositivo lleva consigo información que puede ser útil para identificar sus características. La información que describe al dispositivo se encuentra asociada con el canal de control. Esta información se divide en tres categorías:

  - **Estándar:** Esta es la información cuya definición es común a todos los dispositivos USB e incluye

elementos como la identificación del fabricante, la clase, la gestión de energía.

- **Clase:** La definición de esta información varía dependiendo del aparato. Es una clasificación de los dispositivos en cuanto a sus prestaciones. La especificación USB provee de muchas clases para facilitar la vida al desarrollador de dispositivos, las clases más utilizadas con Microcontroladores son:
  - **HID (Human Interface Device)**: ejemplos de dispositivos que utilizan esta clase son: teclados, ratones, pantallas táctiles, joystick, etc. Velocidad low-speed (64 KB/s de velocidad máxima), tipos de transferencias soportadas: de control y de Interrupción. Una característica interesante al utilizar esta clase es que no se necesita instalar un driver específico en el Sistema Operativo, se utiliza uno estándar que ya está incluido en el sistema.
  - **MSD (Mass Storage Device Class)**: Como su propio nombre indica para dispositivos de almacenamiento masivo como discos duros, memorias flash, cámaras digitales, dispositivos ópticos externos como lectores y grabadoras de CD y DVD, etc. Esta clase se puede utilizar solo en dispositivos que soporten velocidades Full y High Speed. El tipo de transferencias utilizadas es Bulk o una combinación formada por transferencias del tipo Control, Bulk y Interrupt. Microchip tiene notas de aplicación sobre esta clase como la **AN1003**, CCS también implementa ejemplos sobre esta clase. No se necesita la

instalación de un driver específico, se utilizan drivers genéricos instalados ya en los Sistemas Operativos, en Windows se utiliza el driver llamado usbstor.sys ubicado en C:\Windows\Sistem32\drivers.

- **CDC (Communications Device Class)**: Un ejemplo de dispositivo que utiliza esta clase son los Modems, La velocidad máxima al utilizar esta clase será de 80 kBytes/s y el tipo de transferencias soportadas son del tipo interrupción y Bulk. Utiliza también driver estándar incluidos ya en el sistema operativo, según el sistema operativo utilizado precisará o no de la instalación del archivo .INF, cuando utilizamos esta clase en el PC nos creará un puerto serie virtual y la comunicación entre el dispositivo y la aplicación de escritorio se hará a través de el al igual que se haría con un puerto serie físico, esto supone una gran ventaja a la hora de diseñar la aplicación de escritorio, ya que cualquier IDE de programación sea del lenguaje que sea, dispone de un componente o librería que permite el acceso al puerto serie fácilmente.
- **Custom Class**: Se utiliza cuando el dispositivo no se asemeja a las características de ninguno de los miembros pertenecientes a otras clases. un ejemplo de dispositivo que utiliza esta clase es el ICD2 o ICD3 de Microchip.

- **USB Vendor:** El fabricante del periférico puede poner aquí cualquier información deseada.

El software del host es capaz de determinar el tipo de dispositivo conectado haciendo uso de esta información y de un direccionamiento individual. Todos los dispositivos USB son accedidos por una dirección USB que es asignada dinámicamente cuando se conecta, asignándole también un número. Cada aparato soporta además uno o más canales a través de los cuales el host puede comunicarse con el dispositivo. Una vez ha sido reconocido e identificado el dispositivo, el software del host puede hacer que los drivers del dispositivo apropiados obtengan el control del nuevo dispositivo conectado.

Cuando desconectamos el dispositivo, la dirección puede ser reutilizada para el próximo dispositivo conectado.

- **HUBS:** Los hubs son un elemento clave en la arquitectura plug and play del USB. Un bus tradicional puede ser dividido en segmentos de bus individuales conectados por puentes. Cada segmento tiene un número limitado de puntos de conexión debido a la limitada energía y la carga del bus. En la arquitectura USB, el número de puntos de conexión de dispositivos se expande añadiendo dispositivos únicos llamados **HUBs** o concentradores. Estos dispositivos expanden la arquitectura del USB de dos formas:

  - Incrementando los puntos de conexión a través de puertos adicionales.
  - Proporcionando energía a los dispositivos al expandir el bus.

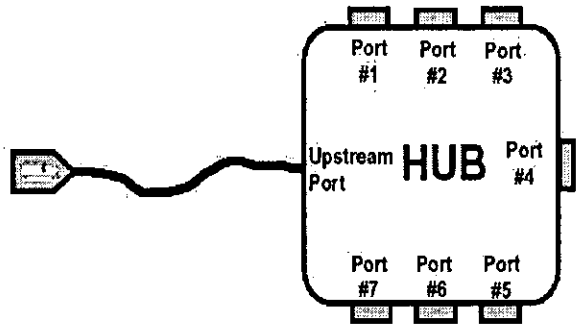


Figura 3. 19 Hub típico

Los HUBs sirven para simplificar la conectividad USB desde la perspectiva del usuario y proporcionar robustez y complejidad a un precio relativamente bajo. Según esta arquitectura, se pueden expandir el bus acoplando hubs adicionales, interconectando dichos hubs mediante enlaces. Cada hub convierte un punto simple de conexión en múltiples puntos, donde estos puntos de conexión se les llaman puertos.

- **Funciones:** Una función es un dispositivo USB que es capaz de transmitir y recibir datos o información de control sobre el bus. Típicamente se implementa como un periférico separado con un cable que se conecta en un puerto del Hub. Sin embargo hay una gran flexibilidad a la hora de construir dispositivos. Una función simple pueden dar una funcionalidad simple (un micrófono, unos altavoces...) o puede estar compuesto en distintos tipos de funcionalidad, como unos altavoces con un panel LCD. Este tipo de dispositivos se les llama función múltiples o ***composite device*** (dispositivo compuesto).

  Otra forma de construir productos con múltiples funciones es creando un ***compound device*** (que significa también dispositivo compuesto). Este es el término usado cuando un Hub está acoplado junto a múltiples dispositivos USB dentro de un mismo paquete. El usuario verá una sola unidad en el extremo del cable, pero internamente tiene un hub y varios dispositivos. Este tipo de "paquetes" tienen una

dirección de bus para cada uno de los componentes, en contraposición a los ***composite devices*** que tienen una única dirección.

Un buen ejemplo de un dispositivo compuesto sería un teclado USB que tuviera una conexión adicional para ratón. A pesar de que el teclado es un periférico, en este caso se le puede acoplar un ratón y por supuesto se necesitaría de un hub interno en el teclado para que esto pueda funcionar.

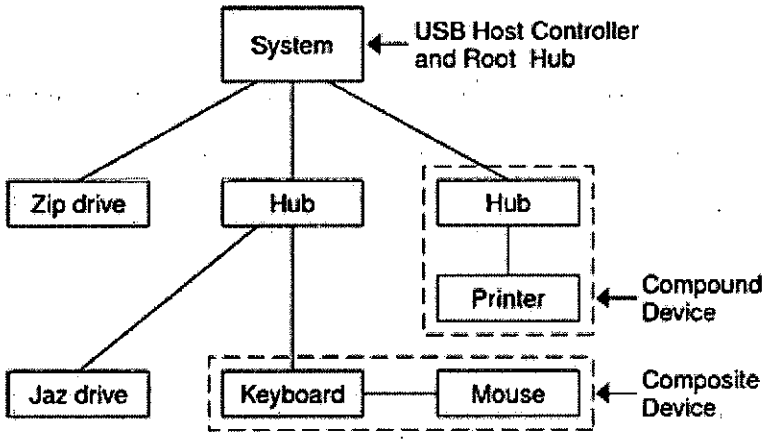


Figura 3. 20 Ejemplo de un dispositivo compuesto en el bus US

Cada función contiene la información sobre la configuración que describe su capacidad y requisitos en cuestión de recursos. Antes de que una función pueda ser usado debe ser configurado por el host.

3.6.3.2. Capa Lógica

El punto de vista lógico presenta capas y abstracciones que son relevantes para los distintos diseñadores e implementadores. La arquitectura lógica describe como unir el hardware del dispositivo USB a un driver del dispositivo en el host para que tenga el comportamiento que el usuario final desea.

La vista lógica de esta conexión es la mostrada en el esquema siguiente. En el podemos ver como el host proporciona conexión al dispositivo, donde esta conexión es a través de un simple enlace USB.

La mayoría de los demás buses como PCI, ISA, etc proporcionan múltiples conexiones a los dispositivos y los drivers lo manipulan mediante algunas combinaciones de estas conexiones (I/O y direcciones de memoria, interrupciones y canales DMA).

Físicamente el USB tiene sólo un cable simple de bus que es compartido por todos los dispositivos del bus. Sin embargo, desde el punto de vista lógico cada dispositivo tiene su propia conexión punto a punto al host.

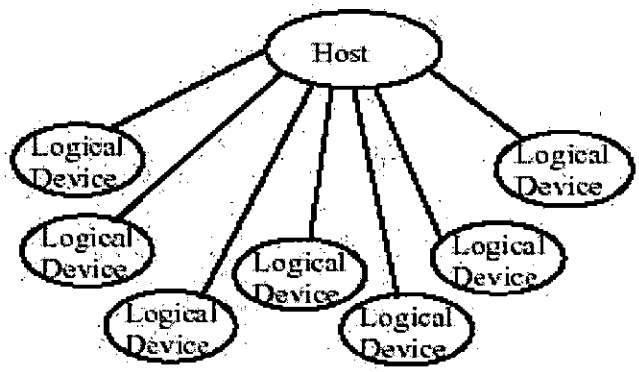


Figura 3. 21 Vista del bus USB desde una capa Lógica

Si lo miramos desde el punto de vista lógico, los hubs son también dispositivos pero no se muestran en la figura para la simplificación del esquema.

Aunque la mayoría de las actividades de los hosts o de los dispositivos lógicos usan esta perspectiva lógica, el host mantiene el conocimiento de la topología física para dar soporte al proceso de desconexión de los hubs. Cuando se quita un hub, todos los dispositivos conectados a él son quitados también de la vista lógica de la topología.

3.6.3.3. Relación "Software del cliente-función"

A pesar de que la topología física y lógica del USB refleja la naturaleza de compartición del bus, la manipulación del interfaz de una función USB por parte del software del cliente (CSw) se presenta con una vista distinta.

El software del cliente para las funciones USB debe usar el interfaz de programación software USB para manipular sus funciones en contraposición de las que son manipuladas directamente a través de la memoria o los accesos I/O como pasa con otros buses (PCI,EISA,PCMCIA,...). Durante esta operación, el software del cliente debería ser independiente a otros dispositivos que puedan conectarse al USB.

3.6.4. Flujo de datos

Un dispositivo USB desde un punto de vista lógico hay que entenderlo como una serie de ***endpoints***, a su vez los ***endpoints*** se agrupan en conjuntos que dan lugar a interfaces, las cuales permiten controlar la función del dispositivo.

Como ya se ha visto la comunicación entre el host y un dispositivo físico USB se puede dividir en tres niveles o capas. En el nivel más bajo el controlador de host USB se comunica con la interfaz del bus utilizando el cable USB, mientras que en un nivel superior el software USB del sistema se comunica con el dispositivo lógico utilizando la tubería de control por defecto (*"Default Control Pipe"*). En lo que al nivel de función se refiere, el software cliente establece la comunicación con las interfaces de la función a través de tuberías asociadas a endpoints.

3.6.4.1. Endpoints y direcciones del dispositivo

Los ***endpoints*** (puntos finales) pueden ser descritos como fuentes o suministros de datos y existen solo en dispositivos USB. Los datos almacenados en un punto final se pueden recibir desde el host o en espera de ser enviado al host. Cada dispositivo USB está compuesto por una colección de endpoints independientes, y una dirección única asignada por el sistema en tiempo de conexión de forma dinámica. A su vez cada endpoint dispone de un identificador único dentro del dispositivo al que pertenece, a este identificador se le conoce como número de endpoint y viene asignado de fábrica. Cada endpoint tiene una determinada orientación de flujo de datos. La combinación de

dirección, número de endpoint y orientación, permite referenciar cada endpoint de forma inequívoca. Cada endpoint es por si solo una conexión simple que soporta un flujo de datos en una única dirección, bien de entrada o bien de salida. Cada endpoint se caracteriza por:

- Frecuencia de acceso al bus requerida
- Ancho de banda requerido
- Numero de endpoint
- Tratamiento de errores requerido
- Máximo tamaño de paquete que el endpoint puede enviar o recibir
- Tipo de ***transferencia*** para el endpoint
- La orientación en la que se transmiten los datos

Existen dos endpoints especiales que todos los dispositivos deben tener, los endpoints con número 0 de entrada y de salida, que deben de implementar un método de control por defecto al que se le asocia la tubería de control por defecto. Estos endpoints están siempre accesibles mientras que el resto no lo estarán hasta que no hayan sido configurados por el host.

3.6.4.2. Pipes o Tuberías

Una tubería USB es una asociación entre uno o dos endpoints en un dispositivo, y el software en el host. Las tuberías permiten mover datos entre software en el host, a través de un buffer, y un endpoint en un dispositivo. Hay dos tipos de tuberías:

- **Stream:** Los datos se mueven a través de la tubería sin una estructura definida. No necesita que los datos se transmitan con una cierta estructura. Las tuberías stream son siempre unidireccionales y los datos se transmiten de forma secuencial: *"first in, first out"* denominado comúnmente como FIFO. Están pensadas para interactuar con un único cliente, por lo que no se mantiene ninguna política de sincronización entre

múltiples clientes en caso de que así sea. Un stream siempre se asocia a un único endpoint en una determinada orientación.

- **Mensaje:** Los datos se mueven a través de la tubería utilizando una estructura USB. A diferencia de lo que ocurre con los streams, en los mensajes la interacción de la tubería con el endpoint consta de tres fases. Primero se realiza una petición desde el host al dispositivo, después se transmiten los datos en la dirección apropiada, finalmente un tiempo después se pasa a la fase estado. Para poder llevar a cabo este paradigma es necesario que los datos se transmitan siguiendo una determinada estructura.

  Las tuberías de mensajes permiten la comunicación en ambos sentidos, de hecho la tubería de control por defecto es una tubería de mensajes.
  El software USB del sistema se encarga de que múltiples peticiones no se envíen a la tubería de mensajes concurrentemente. Un dispositivo ha de dar únicamente servicio a una petición de mensaje en cada instante por cada tubería de mensajes.
  Una tubería de mensajes se asocia a un par de endpoints de orientaciones opuestas con el mismo número de endpoint.

Además una tubería se caracteriza por:

- Demanda de acceso al bus y uso del ancho de banda
- Un tipo de ***transferencia***
- Las características asociadas a los endpoints.

Como ya se ha comentado, la tubería que está formada por dos endpoints con número cero se denomina tubería de control por defecto. Esta tubería está siempre disponible una vez se ha conectado el dispositivo y ha recibido un reseteo del bus. El resto de tuberías aparecen después que se configure el dispositivo. La tubería de control

por defecto es utilizada por el software USB del sistema para obtener la identificación y requisitos de configuración del dispositivo, y para configurar al dispositivo.

El software cliente normalmente realiza peticiones para transmitir datos a una tubería vía IRPs (I/O Request Packet) y entonces, o bien espera, o bien es notificado de que se ha completado la petición. El software cliente puede causar que una tubería devuelva todas las IRPs pendientes. El cliente es notificado de que una IRP se ha completado cuando todas las transacciones del bus que tiene asociadas se han completado correctamente, o bien porque se han producido errores.

Una IRP puede necesitar de varias tandas para mover los datos del cliente al bus. La cantidad de datos en cada tanda será el tamaño máximo de un paquete excepto el último paquete de datos que contendrá los datos que faltan. De modo que un paquete recibido por el cliente que no consiga llenar el buffer de datos de la IRP puede interpretarse de diferentes modos en función de las expectativas del cliente, si esperaba recibir una cantidad variable de datos considerará que se trata del último paquete de datos, sirviendo pues como delimitador de final de datos, mientras que si esperaba una cantidad específica de datos lo tomará como un error.

3.6.4.3. Frame y microframes

USB establece una unidad de tiempo base equivalente a 1 milisegundo denominada frame y aplicable a buses de velocidad media o baja, en alta velocidad se trabaja con microframes, que equivalen a 125 microsegundos. Los (micro) frames no son más que un mecanismo del bus USB para controlar el acceso a este, en función del tipo de transferencia que se realice. En un (micro) frame se pueden realizar diversas transacciones de datos.

por defecto es utilizada por el software USB del sistema para obtener la identificación y requisitos de configuración del dispositivo, y para configurar al dispositivo.

El software cliente normalmente realiza peticiones para transmitir datos a una tubería vía IRPs (I/O Request Packet) y entonces, o bien espera, o bien es notificado de que se ha completado la petición. El software cliente puede causar que una tubería devuelva todas las IRPs pendientes. El cliente es notificado de que una IRP se ha completado cuando todas las transacciones del bus que tiene asociadas se han completado correctamente, o bien porque se han producido errores.

Una IRP puede necesitar de varias tandas para mover los datos del cliente al bus. La cantidad de datos en cada tanda será el tamaño máximo de un paquete excepto el último paquete de datos que contendrá los datos que faltan. De modo que un paquete recibido por el cliente que no consiga llenar el buffer de datos de la IRP puede interpretarse de diferentes modos en función de las expectativas del cliente, si esperaba recibir una cantidad variable de datos considerará que se trata del último paquete de datos, sirviendo pues como delimitador de final de datos, mientras que si esperaba una cantidad específica de datos lo tomará como un error.

3.6.4.3. Frame y microframes

USB establece una unidad de tiempo base equivalente a 1 milisegundo denominada frame y aplicable a buses de velocidad media o baja, en alta velocidad se trabaja con microframes, que equivalen a 125 microsegundos. Los (micro) frames no son más que un mecanismo del bus USB para controlar el acceso a este, en función del tipo de transferencia que se realice. En un (micro) frame se pueden realizar diversas transacciones de datos.

3.6.4.4. Tipos de transferencias

La interpretación de los datos que se transmitan a través de las tuberías, independientemente de que se haga siguiendo o no una estructura USB definida, corre a cargo del dispositivo y del software cliente. No obstante, USB proporciona cuatro tipos de transferencia de datos sobre las tuberías para optimizar la utilización del bus en función del tipo de servicio que ofrece la función. Estos cuatro tipos son:

3.6.4.4.1. Transferencias de control

Es el único tipo de transferencia que utiliza tuberías de mensajes, soporta por lo tanto comunicaciones de tipo configuración/comando/estado entre el software cliente y su función. Una transferencia de tipo control se compone de una transacción de setup del host a la función, cero o más transacciones de datos en la dirección indicada en la fase de setup, y por último una transacción de estado de la función al host. La transacción de estado devolverá éxito cuando el endpoint haya completado satisfactoriamente la operación que se había solicitado.

Por lo tanto este tipo de transferencia está pensado para configurar, obtener información, y en general para manipular el estado de los dispositivos. El tamaño máximo de datos que se transmiten por el bus viene determinado por el endpoint. En dispositivos de velocidad media los posibles tamaños máximos son de 8, 16, 32 o 64 bytes, en velocidad baja el tamaño es de 8 bytes y en velocidad alta 64 bytes. El porcentaje de (micro) frame utilizado ronda el 30% en velocidad baja, 5% en velocidad media y el 2% en alta.

El endpoint puede estar ocupado durante la fase de envío de datos y la fase de estado, en esos casos el endpoint indica al host que se encuentra ocupado, invitando al host a

intentarlo más tarde. Si el endpoint recibe un mensaje de setup y se encontraba en mitad de una transferencia de control, aborta la transferencia actual y pasa a la nueva que acaba de recibir. Normalmente el host no inicia una nueva transferencia de control con un endpoint hasta que no ha acabado la actual, si bien debido a problemas de transmisión el host puede considerar que se han producido errores y pasar a la siguiente. USB proporciona detección y recuperación, vía retransmisión, de errores en las transferencias de control.

3.6.4.4.2. Transferencias isócronas

Hacen uso de tuberías stream. Garantiza un acceso al bus USB con una latencia limitada, asegura una transmisión constante de los datos a través de la tubería siempre y cuando se suministren datos, además en caso de que la entrega falle debido a errores no se intenta reenviar los datos.

USB limita el máximo tamaño de datos para los endpoints con tipo de transferencia isócrona a 1023 bytes para los endpoints de velocidad media y 1024 bytes para velocidad alta. De hecho las transferencias isócronas solo se pueden usar en dispositivos de velocidad alta o media. En función de la cantidad de datos que se estén transmitiendo en un momento dado, en velocidad media el porcentaje de frame utilizado puede variar desde un 1% hasta un 69%, mientras que el porcentaje de microframe utilizado en velocidad alta varía entre un 1% y un 41%.

3.6.4.4.3. Trasferencias de interrupción

Utiliza tuberías stream. Este tipo de transferencia está diseñado para servicios que envían o reciben datos de forma infrecuente. Esta trasferencia garantiza el máximo servicio para la tubería durante el periodo en el que envía. En caso de

error al enviar los datos se reenvían en el próximo periodo de envío de datos.

El tamaño de paquete de datos máximo es de 1024 bytes para alta velocidad, 64 bytes para velocidad media y 8 bytes para baja velocidad. En ningún caso se precisa que los paquetes sean de tamaño máximo, es decir, no es necesario rellenar los paquetes que no alcancen el máximo.

Cuando en una transferencia de interrupción se necesite transmitir más datos de los que permite el paquete máximo, todos los paquetes a excepción del último paquete deben de tener el tamaño máximo. De modo que la transmisión de un paquete se ha llevado a cabo cuando se ha recibido la cantidad exacta esperada o bien, se ha recibido un paquete que no alcanza el tamaño máximo. El porcentaje de (micro) frame utilizado ronda el 13% en velocidad baja y el 2.5% en velocidad media, mientras que en velocidad alta para cantidades similares utilizadas para obtener los anteriores porcentajes se obtienen resultados del 1%, pero para cantidades muy superiores se puede llegar a una utilización del 42%.

3.6.4.4.4. Transferencias de volumen ("Bulk")

Hace uso de tuberías stream. Está diseñado para dispositivos que necesitan transmitir grandes cantidades datos en un momento determinado sin importar mucho el ancho de banda disponible en ese momento. Esta transferencia garantiza el acceso al USB con el ancho de banda disponible, además en caso de error se garantiza el reenvío de los datos. Por lo tanto este tipo de transferencia garantiza la entrega de los datos pero no un determinado ancho de banda o latencia.

El tamaño máximo de paquete de datos para velocidad media es de 8, 16, 32 o 64 bytes, mientras que en velocidad alta

es de 512 bytes. Los dispositivos de velocidad baja no disponen de endpoints con este tipo de transferencia. No es necesario que los paquetes se rellenen para alcanzar el tamaño máximo. El porcentaje de frame utilizado en velocidad media en función del número de bytes enviados varía entre el 1% y el 5%, mientras que el porcentaje de microframe en velocidad alta varía entre un 1% y un 5%, eso sí, teniendo en cuenta mayor cantidad de datos

3.6.5. Capa de protocolo

La forma en la que las secuencias de bits se transmiten en USB es la siguiente; primero se transmite el bit menos significativo, después el siguiente menos significativo y así hasta llegar al bit más significativo. Cuando se transmite una secuencia de bytes se realiza en formato ***"little-endian"***, es decir del byte menos significativo al byte más significativo.

En la transmisión se envían y reciben paquetes de datos, cada paquete de datos viene precedido por un campo Sync y acaba con el delimitador EOP, todo esto se envía codificado además de los bits de relleno insertados. En este punto cuando se habla de datos se refiere a los paquetes sin el campo Sync ni el delimitador EOP, y sin codificación ni bits de relleno.

El primer campo de todo paquete de datos es el campo PID. El PID indica el tipo de paquete y por lo tanto, el formato del paquete y el tipo de detección de errores aplicado al paquete. En función de su PID podemos agrupar los diferentes tipos de paquetes en cuatro clases:

- Token
- Data
- Handshake
- Special

PID Type	PID Name	PID[3:0]*	Descripción
Token	OUT	0001B	Transacción host-to-function. Lleva dirección + número de endpoint
	IN	1001B	Transacción function-to-host. Lleva dirección + número de endpoint
	SOF	0101B	Marca de Start-of-Frame y número de frame
	SETUP	1101B	Transacción host-to-function para SETUP en un Control Pipe. Lleva Dirección + número de endpoint
Data	DATA0	0011B	Paquete de datos para PID para
	DATA1	1011B	Paquete de datos para PID impar
Handshake	ACK	0010B	El receptor acepta un paquete de datos libre de error
	NAK	1010B	El dispositivo Rx no puede aceptar datos o el dispositivo Tx no puede enviar datos
	STALL	1110B	El Endpoint se halteó o no está soportado el requerimiento a un control pipe.
Special	PRE	1100B	Preámbulo enviado por el Host. Habilita tráfico downstream en el bus para dispositivos low-speed.

Figura 3. 22. Clasificación de paquetes según su tipo de PID

Los paquetes se dividen en campos, y cada uno de estos diferentes campos tiene un formato que describiremos a continuación.

3.6.5.1. Campo identificador de paquete (PID)

Es el primer campo que aparece en todo paquete. El PID indica el tipo de paquete, y por tanto el formato del paquete y el tipo de detección de error aplicado a este. Se utilizan cuatro bits para la codificación del PID, sin embargo el campo PID son ocho bits, que son los cuatro del PID seguidos del complemento a 1 de esos cuatro bits. Estos últimos cuatro bits sirven de confirmación del PID. Si se recibe un paquete en el que los cuatro últimos bits no son el complemento a 1 del PID, o el PID es desconocido, se considera que el paquete está corrupto y es ignorado por el receptor.

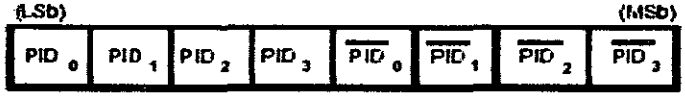


Figura 3. 23 Formato del campo PID

3.6.5.2. Campo dirección

Este campo indica la función, a través de la dirección, que envía o es receptora del paquete de datos. Se utilizan siete bits, de lo cual se deduce que hay un máximo de 128 direcciones.

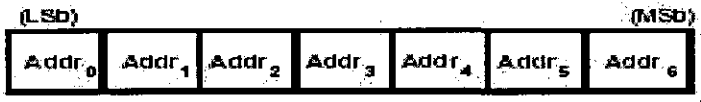


Figura 3. 24 Formato del campo de dirección

3.6.5.3. Campo Endpoint

Se compone de cuatro bits e indica el número de *"endpoint"* al que se quiere acceder dentro de una función, como es lógico este campo siempre sigue al campo dirección.

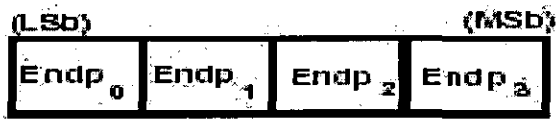


Figura 3. 25 Formato del campo Endpoint

3.6.5.4. Campo número de frame

Es un campo de 11 bits que es incrementado por el host cada (micro) frame en una unidad. El máximo valor que puede alcanzar es el 7FFH, si se vuelve a incrementar pasa a cero.

3.6.5.5. Campo de datos

Los campos de datos pueden variar de 0 a 1024 bytes. En el dibujo se muestra el formato para múltiples bytes.

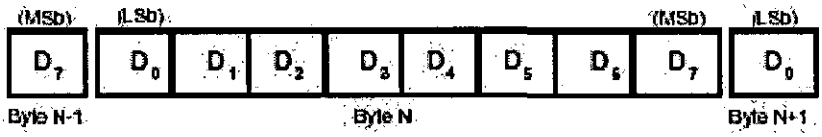


Figura 3. 26 Formato del campo de datos

3.6.5.6. Cyclic Redundancy Checks *(CRC)*

El CRC se usa para proteger todos los campos no PID de los paquetes de tipo token y de datos. El CRC siempre es el último campo y se genera a partir del resto de campos del paquete, a excepción del campo PID. El receptor al recibir el paquete comprueba si se ha generado de acuerdo a los campos del paquete, si no es así, se considera que alguno o más de un campo están corruptos, en ese caso se ignora el paquete.

El CRC utilizado detecta todos los errores de un bit o de dos bits. El campo de CRC es de cinco bits para los paquetes de tipo IN, SETUP, OUT, PING y SPLIT.

3.6.5.7. Paquetes de tipo token

Un token está compuesto por un PID que indica si es de tipo IN, OUT o SETUP. El paquete especial de tipo PING también tiene la misma estructura que token. Después del campo PID viene seguido de un campo dirección y un campo endpoint, por último hay un campo CRC de 5 bits.

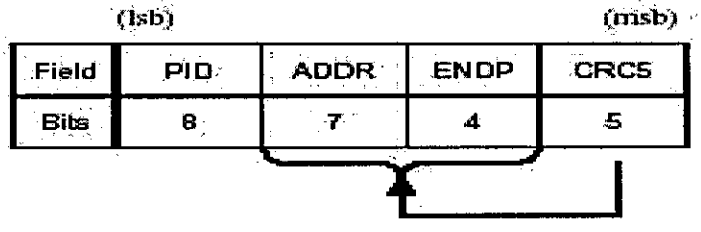


Figura 3. 27 Formato del paquete tipo Token

En los paquetes OUT y SETUP esos campos identifican al endpoint que va a recibir el paquete datos que va a continuación. En los paquetes IN indican el endpoint que debe transmitir un paquete de datos. En el caso de los paquetes PING hacen referencia al endpoint que debe responder con un paquete *"handshake"*.

3.6.5.8. Paquete inicio de frame (SOF)

Estos paquetes son generados por el host cada un milisegundo en buses de velocidad media y cada 125 microsegundos para velocidad alta. Este paquete está compuesto por un campo número de frame y un campo de CRC de 5 bits.

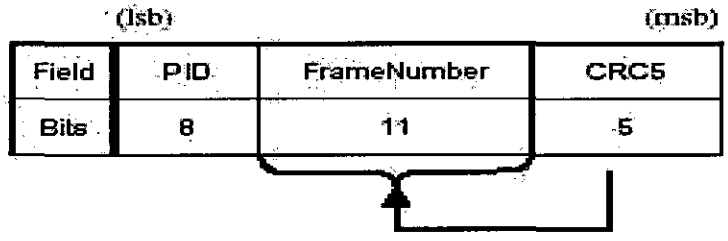


Figura 3. 28 Formato del paquete SOF

3.6.5.9. Paquete de datos

Este paquete está compuesto por cero o más bytes de datos seguido de un campo de CRC de 16 bits. Existen cuatro tipos de paquetes de datos: DATA0, DATA1, DATA2 y MDATA. El número máximo de bytes de datos en velocidad baja es de ocho bytes, en media de 1023 bytes y en alta de 1024 bytes. El número de bytes de datos ha de ser entero.

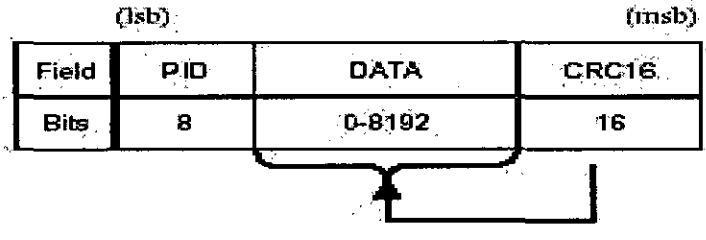


Figura 3. 29 Formato del paquete de datos

3.6.5.10. Paquetes *"Handshake"*

Los paquetes *"handshake"*, en castellano apretón de manos, se utilizan para saber el estado de una transferencia de datos, indicar la correcta recepción de datos, aceptar o rechazar comandos, control de

flujo, y condiciones de parada. El único campo que contiene un paquete de este tipo es el campo PID.

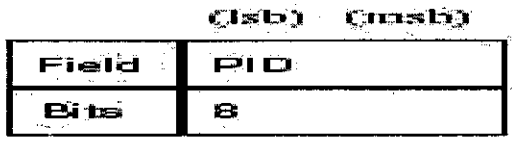


Figura 3. 30 Formato del paquete Handshake

Existen cuatro paquetes de tipo *"Handshake"* además de uno especial:

- **ACK:** Indica que el paquete de datos ha sido recibido y decodificado correctamente. ACK sólo es devuelto por el host en las transferencias IN y por una función en las transferencias OUT, SETUP o PING.
- **NAK:** Indica que una función no puede aceptar datos del host (OUT) o que no puede transmitir datos al host (IN). También puede enviarlo una función durante algunas fases de transferencias IN, OUT o PING. Por último se puede utilizar en el control de flujo indicando disponibilidad. EL host nunca puede enviar este paquete.
- **STALL:** Puede ser devuelto por una función en transacciones que intervienen paquetes de tipo IN, OUT o PING. Indica que una función es incapaz de transmitir o enviar datos, o que una petición a una tubería control no está soportada. El host no puede enviar bajo ninguna condición paquetes STALL.
- **NYET:** Sólo disponible en alta velocidad es devuelto como respuesta bajo dos circunstancias. Como parte del protocolo PING, o como respuesta de un hub a una transacción SPLIT indicando que la transacción de velocidad media o baja aún no ha terminado, o que el hub no está aún habilitado para realizar la transacción.
- **ERR:** De nuevo sólo disponible en alta velocidad y de nuevo formando parte del protocolo PING, permite a un hub de alta velocidad indicar que se ha producido un error en un bus de media o baja velocidad.

3.6.6. Especificaciones eléctricas

3.6.6.1. Identificación de la velocidad del dispositivo

Para poder iniciar cualquier tipo de transacción cuando se conecta el dispositivo al host, es necesario que este conozca la velocidad a la que trabaja. Con esa finalidad existe un mecanismo a nivel eléctrico. La diferencia entre los dispositivos de velocidad media y los de velocidad baja, es que en velocidad media tiene una resistencia conectada al D+, en velocidad baja la misma resistencia se encuentra en D- y no en D+ como se puede observar en los dibujos.

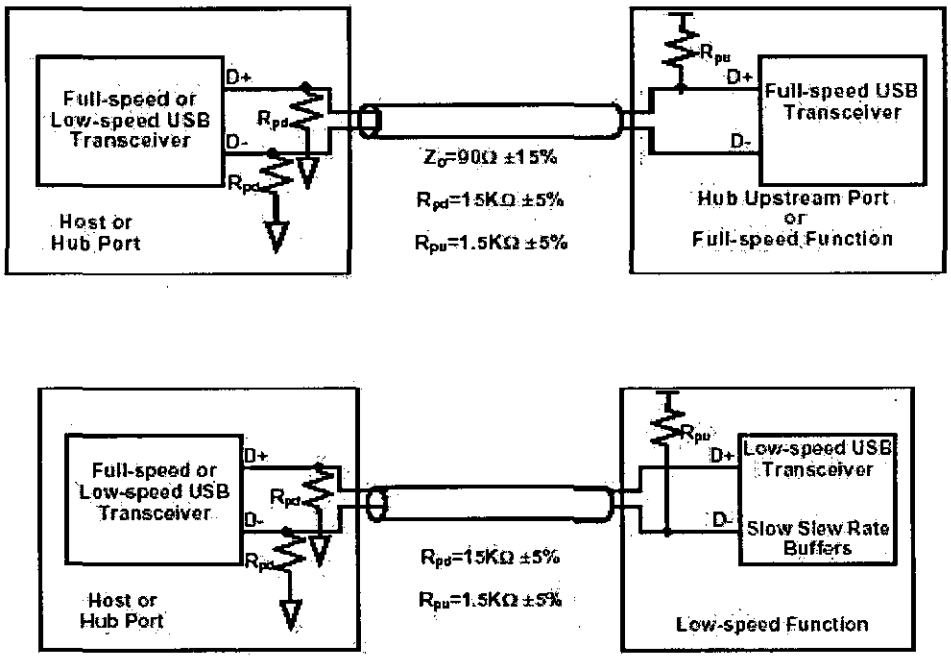


Figura 3. 31 Método de reconocimiento de dispositivos Low Speed y High Speed

De forma que después del reset el estado de reposo de la línea es diferente si se trata de baja o media velocidad. En el caso de dispositivos de alta velocidad lo que se hace es que en un principio se conecta como un dispositivos de velocidad media y más tarde a través de un protocolo se pasa a velocidad alta.

3.6.6.2. Codificación de datos

El USB utiliza la codificación NRZI para la transmisión de paquetes. En esta codificación los "0" se representan con un cambio en el nivel, y por el contrario los "1" se representan con un no cambio en el nivel. De modo que las cadenas de cero producen transiciones consecutivas en la señal, mientras que cadenas de unos produce largos periodos sin cambios en la señal. A continuación un ejemplo:

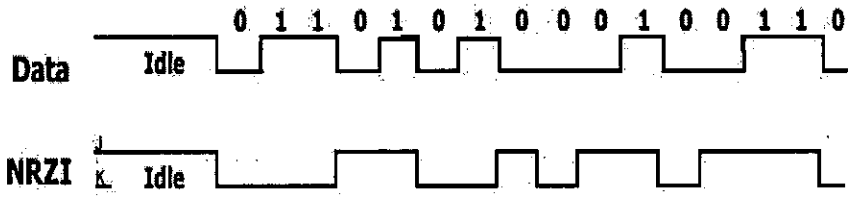


Figura 3. 32 Codificación NZRI para transmisión de paquetes

3.6.6.3. Relleno de bits

Debido a que cadenas de unos pueden producir largos periodos en los que la señal no cambia dando lugar a problemas de sincronización, se introducen los bits de relleno. Cada 6 bits consecutivos a "1" se inserta un bit a "0" para forzar un cambio, de esta forma el receptor puede volverse a sincronizar. El relleno bits empieza con el patrón de señal Sync. El "1" que finaliza el patrón de señal Sync es el primer uno en la posible primera secuencia de seis unos.

En las señales a velocidad media o baja, el relleno de bits se utiliza a lo largo de todo el paquete sin excepción. De modo que un paquete con siete unos consecutivos será considerado un error y por lo tanto ignorado.

En el caso de la velocidad alta se aplica el relleno de bits a lo largo del paquete, con la excepción de los bits intencionados de error usados en EOP a velocidad alta.

3.6.6.4. Sync

Teniendo en cuenta que K y J representan respectivamente nivel bajo y nivel alto, el patrón de señal Sync emitido, con los datos codificados, es de 3 pares KJ seguidos de 2 K para el caso de velocidad media y baja. Para velocidad alta es una secuencia de 15 pares KJ seguidos de 2 K. A continuación el caso de velocidad media y baja:

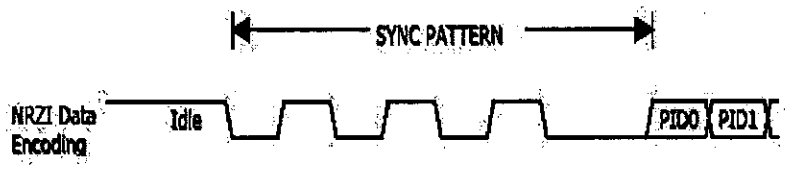


Figura 3. 33 Patrón de sincronismo del campo Sync

El patrón de señal Sync siempre precede al envío de cualquier paquete, teniendo como objetivo que el emisor y el receptor se sincronicen y se preparen para emitir y recibir datos respectivamente. Si partimos de que el estado de reposo de la señal es J, podemos interpretar Sync como una secuencia impar de "0's" y un "1" que se inserta antes de los datos.

3.6.6.5. EOP (*"End Of Packet"*)

A todo paquete le sigue EOP, cuya finalidad es indicar el final del paquete.
En el caso de velocidad media y baja el EOP consiste en que, después del último bit de datos en el cual la señal estará o bien en estado J, o bien en estado K, se pasa al estado SE0 durante el periodo que se corresponde con el ocupado por dos bits, finalmente se transita al estado J que se mantiene durante 1 bit. Esta última transición indica el final del paquete.

En el caso de la velocidad alta se utilizan bits de relleno erróneos, que no están en el lugar correcto, para indicar el EOP.

Concretamente, el EOP sin aplicar codificación consistiría en añadir al final de los datos la secuencia 0111 1111.

3.7. Comunicación USB con PIC C

Si queremos realizar la programación de los microcontroladores PIC en lenguaje como el C, es preciso utilizar un compilador C. Dicho compilador nos genera ficheros en formato Intel-hexadecimal, que es necesario para programar (utilizando un programador de PIC, en nuestro caso un PICKIT 2) un microcontrolador. El compilador de C que vamos a utilizar es el PCW de CCS inc. El compilador C de CCS ha sido desarrollado específicamente para PIC MCU, obteniendo la máxima optimización del compilador con estos dispositivos. Dispone de una amplia librería de funciones predefinidas, comando de procesado y ejemplos. Además, suministra los controladores (drivers) para diversos dispositivos como LCD, convertidores AD, relojes en tiempo real, EEPROM serie, etc.

El CCS es C estándar y, además de las directivas estándar ***(#include, etc)***, suministra unas directivas específicas para PIC ***(#device, etc)***; además incluye funciones específicas ***(bit_set(), etc)***. A su vez, el compilador lo integraremos en un entorno de desarrollo integrado (IDE) que nos va a permitir desarrollar todas y cada una de las fases que se compone un proyecto, desde la edición hasta la compilación pasando por la depuración de errores. La última fase, a excepción de la depuración y retoques del hardware finales, será programar el PIC.

CCS suministra librerías para comunicar PIC con el PC, utilizando el bus USB mediante periféricos internos (familia PIC18F2550 o el PIC 16C765) o mediante dispositivos externos del PIC (del tipo USBN9603).

Funciones relevantes:

- **usb_init ():** Inicializa el hardware USB. Entonces tendrá que esperar en un bucle infinito por el periférico USB que se conecta al bus (pero eso no quiere decir que se ha dividido por el PC). Will habilitar y utilizar la interrupción USB.

- **usb_init_cs ():** El mismo usb_init (), pero no espera a que el dispositivo esté conectado al bus. Esto es útil si el dispositivo no está alimentado por bus y pueden funcionar sin una conexión USB.
- **usb_task ():** Si usa el sentido de conexión y la usb_init_cs () para la inicialización, entonces usted debe llamar periódicamente esta función para mantener un ojo en el pin sentido conexión. Cuando el PIC está conectado al BUS, esta función entonces perpare el periférico USB. Cuando el PIC está desconectado del bus, se restablecerá la pila USB y periférico. Will habilitar y utilizar la interrupción USB.
- **usb_detach ():** Elimina el PIC desde el autobús. Se llamará automáticamente por usb_task () si se pierde la conexión, pero puede ser llamada manualmente por el usuario.
- **usb_attach ():** Se conecta al PIC al bus. Se llamará automáticamente por usb_task () si la conexión, pero puede ser llamada manualmente por el usuario.
- **usb_attached ():** Si se utiliza pines sentido (USB_CON_SENSE_PIN), devuelve TRUE si ese pin es alto. Otras ventas siempre devolverá VERDADERO.
- **usb_enumerated ():** Devuelve TRUE si el dispositivo ha sido enumerado por la PC. Si el dispositivo ha sido enumerado por la PC, que significa que está en modo de operación normal y se puede enviar / recibir paquetes.
- **usb_put_packet:** (Punto final, los datos, len, TGL) Coloca el paquete de datos en el búfer de punto final especificado. Devuelve TRUE si el éxito, FALSE si el buffer está todavía lleno con el último paquete.
- **usb_puts (Punto final, los datos, len, tiempo de espera):** Envía los datos siguientes para el punto final especificado. usb_puts () difiere de usb_put_packet () en que va a enviar mensajes de múltiples paquetes si los datos no caben en un paquete.
- **usb_kbhit (punto final):** Devuelve TRUE si el punto final especificado tiene datos en el búfer de recepción es
- **usb_get_packet (Endpoint, ptr, max):** Lee hasta un máximo de bytes del búfer de extremo especificado y lo guarda en el ptr puntero. Devuelve el número de bytes guardados en ptr.

- **usb_gets (punto final, PTR, máximo, tiempo de espera):** Lee un mensaje desde el punto final especificado. La diferencia usb_get_packet () y usb_gets () es que usb_gets () esperará hasta que un mensaje completo se ha recibido, que un mensaje puede contener más de un paquete. Devuelve el número de bytes recibidos.

Funciones relevantes del CDC:

Un dispositivo USB CDC emular un dispositivo RS-232, y aparecerá en su ordenador como un puerto COM. Las funciones siguen ofrecerle esta interfaz virtual de RS-232/serial

Nota: Cuando se utiliza la biblioteca de CDC, se pueden utilizar las mismas funciones que el anterior, pero no utilice la función de paquetes relacionados, tales como **usb_kbhit (), usb_get_packet (), etc**

- **usb_cdc_kbhit ():** El mismo kbhit (), devuelve TRUE si hay 1 o más caracteres en la búfer de recepción.
- **usb_cdc_getc ():** Al igual que getc (), lee y devuelve un carácter del búfer de recepción. Si no hay datos en el búfer de recepción que esperará indefinidamente hasta que un carácter se ha recibido.
- **usb_cdc_putc (c):** El mismo putc (), envía un carácter. De hecho pone un carácter en el búfer de transmisión, y si la memoria intermedia de transmisión está llena esperará indefinidamente hasta que haya espacio para el personaje.
- **usb_cdc_putc_fast (c):** El mismo usb_cdc_putc (), pero no esperará indefinidamente hasta que haya espacio para el carácter en el búfer de transmisión. En esa situación, el carácter se pierde.
- **usb_cdc_putready ():** Devuelve TRUE si hay espacio en el buffer de transmisión para otro personaje.

Incluir archivos relevantes:

- **pic_usb.h:** Hardware conductor capa para los controladores PIC16C765 familia PICmicro con un periférico USB interno.

- **pic_18usb.h:** Hardware conductor capa para los controladores de la familia PIC18F4550 PICmicro con un periférico USB interno.
- **usbn960x.h:** Hardware conductor capa de la National USBN9603/USBN9604 externo USB periférico. Puede utilizar este periférico externo para añadir USB a cualquier microcontrolador.
- **usb.h:** Definiciones comunes y prototipos usados por el controlador USB
- **usb.c:** La pila USB, que maneja la interrupción USB y USB solicitudes de instalación de punto final 0.
- **usb_cdc.h :** Un conductor que lleva la anterior incluyen archivos para hacer un dispositivo USB CDC, que emula un dispositivo heredado RS232 y se muestra como un puerto COM en el administrador de dispositivo de MS Windows.

3.8. Comunicación USB con Matlab

Para a creación del sistema de comunicaciones entre el microcontrolador y Windows, primeramente hay que introducir en la librería del firmware un apartado que se denomina "***descriptor***". Esta parte de la librería se utiliza para suministrar a Windows la información que necesita en el proceso de enumeración como lo es el VID, PID, ENDPOINT entre otros, del dispositivo que quiere crear una vía de comunicación con el sistema operativo.

Una vez introducido el firmware en el controlador y que conectamos el cable USB a una computadora cualquiera, lo primero que pedirá el sistema operativo es un driver para terminar de configurar el dispositivo. Este driver lo proporciona microchip e ira incluido en el soporte físico del proyecto.

Terminado de instalar el driver (la instalación del driver es la misma forma que la instalación del driver de cualquier dispositivo que se instala en windows) se arrancara Matlab y se podrá ejecutar las correspondientes instrucciones creadas para su uso.

Para el establecimiento de una comunicación entre un dispositivo ya enumerado en windows, es decir, ya instalado o reconocido por windows y un programa aparte se utilizan normalmente unas librerías dinámicas denominadas

librerías DLL o Dinamic Link Librarys. Estas librerías no son más que código compilado. Este código está compuesto por una serie de funciones, datos y otros recursos (bitmaps, tablas, etc.) en código objeto, que son las que se utilizaran para el control del dispositivo. Por tanto, desarrollando estas librerías se puede reagrupar código y usarlo donde se requiera este tipo de funcionalidad. El código de las librerías es añadido al código de la aplicación mediante el proceso denominado LINKING.

Una de las técnicas utilizadas por los compiladores para ligar librerías a una aplicación es la denominada **Static Linking.** En este método todas las librerías y el código objeto de la aplicación son combinados para crear un fichero final en formato HEX utilizado para programar el dispositivo externo; en el caso de este proyecto, un microcontrolador.

Sin embargo, en un sistema multitarea como Windows, se puede ejecutar dos o más copias de una misma aplicación. Y por tanto, serian dos o más copias de una misma aplicación. Y por tanto, serian dos o más copias de una misma aplicación que podrían utilizar una misma librería. Esto implicaría la aparición de un código y datos redundantes en la memoria al mismo tiempo causando un gasto de recursos innecesarios. Además, la carga de trabajo del sistema operativo aumentaría con la carga y descarga de la aplicación en memoria que consumiría más tiempo.

Para suprimir estos problemas, los sistemas multitarea recurren a otra técnica para unir las librerías a las aplicaciones conocidas como **Dynamic-Linking.** Las librerías que emplean este método se denominan *Dynamic-Link-Librarys* y tienen la extensión DLL.

En este caso, microchip proporciona una dll para la comunicación USB con sus microcontroladores de la familia PIC18FXX5X denominada *mpusbapi.dll.* Esta dll está compuesta por una serie de funciones que serán las funciones básicas en las que están basadas las funciones que utilizaremos en Matlab. Para su correcto funcionamiento correcto, se necesita el driver *mchpusb.sys.*

3.8.1. Funciones de la librería MPUSBAPI.DLL

3.8.1.1. MPUSBGETDLLVERSION(VOID)

Lee el nivel de revisión del MPUSAPI.dll. Es un nivel de revisión de 32 bits. Esta función no devuelve la versión del código, no realiza nada con el USB, solo devuelve la versión de la dll en formato hexadecimal de 32 bits.

3.8.1.2. MPUSBGETDEVICECOUNT(pVID_PID)

Devuelve el número de dispositivos con un VID_PID asignado. El campo **"pVID_PID"** es cadena de caracteres del número de identificación asignado.

3.8.1.3. MPUSBOPEN(INSTANCE,pVID_PID,pEP,DWDIR, DWRESERVED)

Devuelve el acceso al pipe del Endpoint con el VID_PID asignado. Todos los pipes se abren con el atributo FILE_FLAG_OVERLAPPED. Esto permite que MPUSBRead, MPUSBWrite y MPUSBReadInt tenga un valor de timeout. El valor del timeout no tiene sentido en una pipe síncrona. Los campos que requiere esta función se detallan a continuación:

- **Instance:** Un número de dispositivo para abrir. Normalmente, se utiliza primero la llamada de **MPUSBGetDeviceCount** para saber cuántos dispositivos hay.

  Es importante entender que el driver lo comparten distintos dispositivos. El número devuelto por el **MPUSBGetDeviceCount** tiene que ser igual o menor que el número de todos los dispositivos actualmente conectados y usando el driver genérico.

Ejemplo:

Si hay tres dispositivos con los siguientes PID_VID conectados:

- Dispositivo tipo 0, VID 0x04d8, PID 0x0001
- Dispositivo tipo 1, VID 0x04d8, PID 0x0002
- Dispositivo tipo 2, VID 0x04d8, PID 0x0003

Si el dispositivo que nos interesa tiene VID=0x04d8 y PID=0x0002 el MPUSBGetDeviceCount devolverá un '1'.

Al llamar la función tiene que haber un mecanismo que intente llamar **MPUSOpen()** desde 0 hasta **MAX_NUM_MPUSB_DEV**. Se tiene que contar el número de llamadas exitosas. Cuando este número sea igual al número devuelto por MPUSBGetDeviceCount, hay que dejar de hacer las llamadas porque no puede haber más dispositivos con el mismo VID_PID.

- **pVID_PID:** String que contiene el PID&VID del dispositivo objetivo. El formato es ***"vid_xxxx&pid_yyyy"***. Donde ***xxxx*** es el valor del VID y el ***yyyy*** el del PID, los dos en hexadecimal.

  Ejemplo:

  Si un dispositivo tiene un VID=0x04d8 y un PID=0x000b, el string de entrada es: ***"vid_04d8&pid_000b"*.**
- **pEP:** String de entrada con el número de Endpoint que se va a abrir. El formato es "\\MCHP_EPz" o "\MCHP_EPz" dependiendo del lenguaje de programación. Donde **"z"** es el número del Endpoint en decimal que va desde el 1 hasta el 16 que designa el canal escogido.
- **dwDir:** Especifica la dirección del Endpoint. Los distintos Endpoint pueden tener dos direcciones, una de envió de datos y otra de recepción de datos. Un mismo canal puede estar abierto para enviar datos como para recibir, sin embargo, el entero o handle que se asocia a ambas formas de comunicar es distinto. Por ejemplo, el Endpoint 1 puede habilitarse como

entrada y salida y tendrá un número asignado para la entrada y un número asignado para la salida. Este argumento es un entero que valdrá 1 para establecer el canal como lectura y 0 para establecerlo como canal de escritura.

- **dwReserved:** Debe ser pasado como argumento. Este argumento en este momento no emplea para nada, está pensado como un argumento que se utilizara en el futuro. En este momento lo único que se envía es 0.

3.8.1.4. MPUSBREAD(HANDLE, PDATA, DWLEN, PLENGTH, DWMILLISECONDS)

- ***handle:*** Identifica la pipe del Endpoint que se va a leer. La pipe unida tiene que crearse con el atributo de acceso MP_READ.
- ***pData:*** Puntero al buffer que recibe el dato leído de la pipe.
- ***dwLen:*** Especifica el número de bytes que hay que leer de la pipe.
- ***pLenght:*** Puntero al número de bytes leídos. MPUSBRead pone este valor a cero antes de cualquier lectura o de chequear un error.
- ***dwMilliseconds:*** Especifica el intervalo de time-out en milisegundos. La función vuelve si transcurre el intervalo aunque no se complete la operación. Si dwMilliseconds=0, la función comprueba los datos de la pipe y vuelve inmediatamente. Si dwMilliseconds es infinito, el intervalo de time-out nunca termina.

3.8.1.5. MPUSBWRITE(HANDLE, PDATA, DWLEN, PLENGTH, DWMILLISECONDS)

- ***handle:*** Identifica la pipe del Endpoint que se va a escribir. La pipe unida tiene que crearse con el atributo de acceso MP_WRITE.
- ***pData:*** Puntero al buffer que contiene los datos que se van a escribir en la pipe.

- ***dwLen:*** Especifica el número de bytes que se van a escribir en la pipe.
- ***pLenght:*** Puntero al número de bytes que se escriben al llamar a esta funcion. MPUSBWITE pone este valor a cero antes de cualquier lectura o de chequear un error.
- ***dwMilliseconds:*** Especifica el intervalo de time-out en milisegundos. La función vuelve si transcurre el intervalo aunque no se complete la operación. Si dwMilliseconds=0, la función comprueba los datos de la pipe y vuelve inmediatamente. Si dwMilliseconds es infinito, el intervalo de time-out nunca termina.

3.8.1.6. MPUSBREADINT(HANDLE, PDATA, DWLEN, PLENGTH, DWMILLISECONDS)

- ***handle:*** Identifica la pipe del Endpoint que se va a leer. La pipe unida tiene que crearse con el atributo de acceso MP_READ.
- ***pData:*** Puntero al buffer que recibe el dato leído de la pipe.
- ***dwLen:*** Especifica el número de bytes que hay que leer de la pipe.
- ***pLenght:*** Puntero al número de bytes leídos. MPUSBRead pone este valor a cero antes de cualquier lectura o de chequear un error.
- ***dwMilliseconds:*** Especifica el intervalo de time-out en milisegundos. La función vuelve si transcurre el intervalo aunque no se complete la operación. Si dwMilliseconds=0, la función comprueba los datos de la pipe y vuelve inmediatamente. Si dwMilliseconds es infinito, el intervalo de time-out nunca termina.

3.8.1.7. MPUSBCLOSE(HANDLE)

Deshabilita los canales que ha sido creados con la función _MPUSBOPEN, de manera que esos números enteros suministrados dejan de identificar canales d comunicación.

3.9. ¿COMO SE INVOCAN FUNCIONES DE UNA LIBRERÍA EN WINDOWS?

Cuando se construye un proyecto de librería se crean 2 ficheros. Uno es el fichero de referencia de la librería (.LIB) y el otro es el fichero en código objeto (.OBJ, .DLL). Lo que se entiende por código objeto no es más que código maquina con las referencias y el mapa de memoria sin resolver, cosas que un ejecutable normal tiene resueltas.

Una vez que son generados los ficheros .LIB y .OBJ, para poder usarlos en lenguaje C o C++ se requiere la creación de lo que se denomina ***fichero de cabecera*** o ***Header file (.h).*** Este fichero se incluirá en el código fuente de la aplicación mediante la instrucción #include y el fichero .LIB se añadirá a los ficheros para ser unidos o *Files to be Linked.* El fichero en código objeto (.OBJ) se colocara en el directorio del código fuente. A veces, el fichero .LIB contiene también el código objeto dependiendo del compilador y por lo tanto no habrá ningún fichero .OBJ.

Una vez generados los ficheros estos se usaran en el método *Static-Linking* comentado anteriormente. Si los ficheros de los que se parten son el .DLL y el .LIB, se someterán al método conocido como *Dynamic-Linking.* Cuando se trata del método *Dynamic-Linking* hay dos formas de acceder a las funciones contenidas en la librería DLL.

La primera es la conocida como el método *Load-Time Linking* o unión en tiempo de carga. En este método, la librería se une a la aplicación usando un .LIB de la misma manera a como sucede en el método *Static-Linking.* La única diferencia es que el fichero DLL asociado al fichero LIB se cargará en tiempo de ejecución.

La segunda forma es conocida como *Run-Time Linking.* En este método, el fichero LIB no será utilizado. La aplicación asume que el fichero DLL esta accesible en el PATH del sistema y utilizara una función Win32 API para acceder a las funciones de la DLL. Estas funciones que emplean para manejar la DLL son *LoadLibrary(), GetProcAddress()* y *FreeLibrary().*

Estas tres funciones son de interés si se va a acceder a las funciones de la librería desde lenguajes de programación en C o C++. Para otros lenguajes de alto nivel, hay otras funciones equivalentes a estas pero que son propias de cada uno. La más importante de las funciones en cuestión es la función *loadlibrary* de Matlab. Básicamente lo que hace es cargar una librería externa en matlab. Lo interesante es que esa librería externa puede ser perfectamente una librería dll como la que se interesa manejar. Su sintaxis en la siguiente:

Loadlibrary ('shrlib','hfile')

En esta instrucción carga las funciones definidas en el archivo de cabecera hfile y localizadas en la librería **shrlib. Shrlib** es el nombre de una Dynamic Link Library, es decir, una dll. Además de la función mencionada es también de interés conocer las funciones:

- *Libfunctionsview*
- *Libpointer*
- *Callib*
- *Libisloaded*
- *Unloadlibrary*

CAPITULO IV

SISTEMA Y EQUIPOS DE COMUNICACION

4.1. Descripción del sistema de comunicación

Nuestro sistema para la toma de datos de aceleración está formado por 3 computadores; Master, Slave y Gateway, los cuales son los encargados de asegurar él envió, recepción e integridad de los paquetes de datos durante la comunicación. Se implementó un protocolo de comunicación entre Master-Gateway-Slave que considera un canal de comunicación real, en donde la data puede perderse en el transcurso de la comunicación por diferentes razones; por lo tanto se adoptan las medidas necesarias para hacer frente a los diferentes casos de error en la comunicación que puedan suscitarse, obtener datos de aceleración y secuencia correctos. A continuacion describimos los 3 elementos del sistema de comunicacion.

4.1.1. Master

El inicio, final o cualquier orden de comunicación son emitidas por el Master, que en nuestro sistema es una PC, la cual ejecuta una interfaz gráfica de usuario (GUIDE), desarrollada para la adquisición y procesamiento de la data de aceleración y configuración del acelerómetro. El sistema es desarrollado con el Software MATLAB, utilizando la comunicación USB de clase *Custom class* con el tipo de transferencia *Bulk Data Transfers* para el intercambio de paquetes con el Gateway.

En lo referente a adquisición de la data de aceleración proveniente del Slave, se consideró todos los posibles errores que pueden ocurrir y se elaboraron dos funciones encargadas de validar la data, una de respuestas a comandos enviados y otra para lo concerniente a data de aceleración, para hacer frente antes perdida de paquetes, paquetes con error, perdida de secuencia de paquetes, ***timeout*** o problemas con pérdidas de conexión. Cuando no se pide data de aceleración o no se envían comandos, la aplicación ejecuta un ***Background Task***, la cual se encarga, cada un segundo, de solicitar

data del nivel de batería del Slave, y asi estar en constante comunicación. Además cuenta con la opción de modificar parametros de nuestro acelerómetro como son el rango de aceleracion, frecuencia de muestreo, colocar al acelerómetro en modo medición o standby en tiempo real.

Para lo que concierne al procesamiento de la data de aceleración para obtener desplazamiento, se vio necesaria la aplicación de técnicas de procesamiento digital, con la finalidad de separar el offset (aceleración de la gravedad) de la data de aceleración dinamica medida en el eje Z. Finalmente aplicamos algoritmos basados en métodos de integración numérica, ya que debemos integrar dos veces la aceleración para poder obtener desplazamiento.

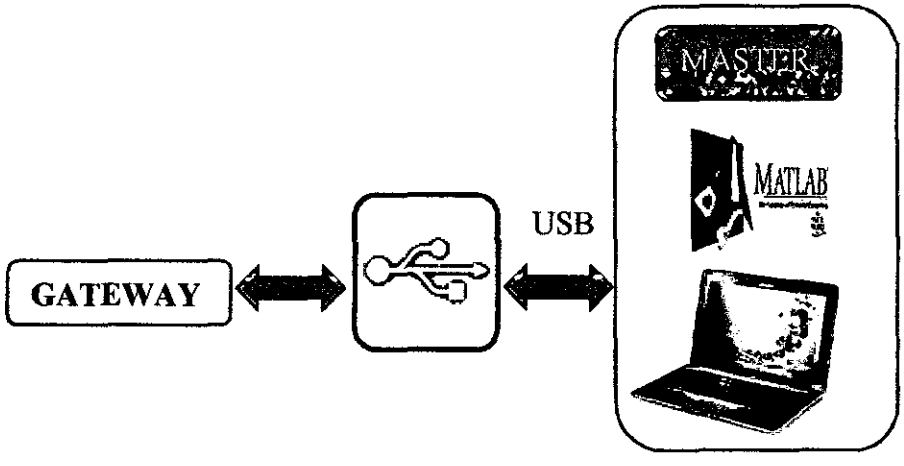


Figura 4. 1 Descripción del dispositivo Master que en nuestro caso es la PC

4.1.2. Slave

Esta tarjeta consta de un acelerómetro, que se comunica con un microcontrolador PIC para la adquisición de datos de aceleración. El Slave a pedido del Master puede modificar los parámetros de rango de aceleracion, número de datos requeridos y cuando iniciar o detener la lectura de datos de aceleración. El Slave también envía al Master datos de voltaje de la bateria que alimenta a la tarjeta Slave. El envío de esta data se da mediante comunicación inalámbrica Bluetooth al Gateway para que este valide y envié la data al Master.

data del nivel de batería del Slave, y asi estar en constante comunicación. Además cuenta con la opción de modificar parametros de nuestro acelerómetro como son el rango de aceleracion, frecuencia de muestreo, colocar al acelerómetro en modo medición o standby en tiempo real.

Para lo que concierne al procesamiento de la data de aceleración para obtener desplazamiento, se vio necesaria la aplicación de técnicas de procesamiento digital, con la finalidad de separar el offset (aceleración de la gravedad) de la data de aceleración dinamica medida en el eje Z. Finalmente aplicamos algoritmos basados en métodos de integración numérica, ya que debemos integrar dos veces la aceleración para poder obtener desplazamiento.

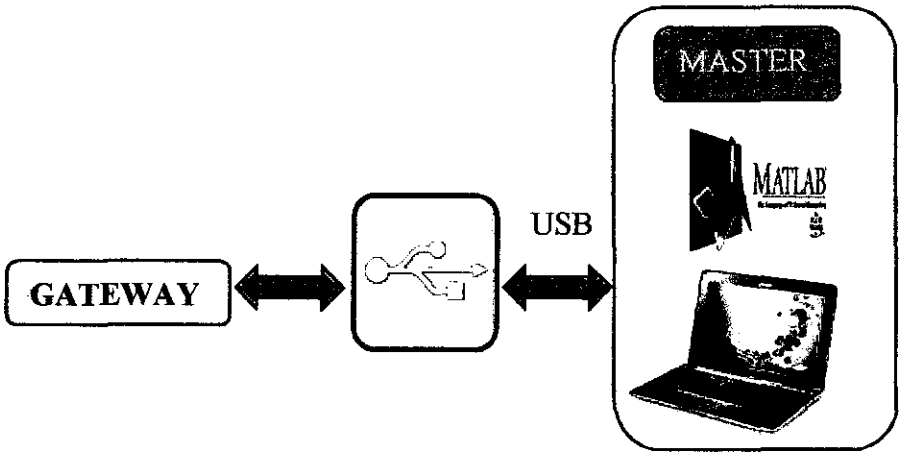


Figura 4. 1 Descripción del dispositivo Master que en nuestro caso es la PC

4.1.2. Slave

Esta tarjeta consta de un acelerómetro, que se comunica con un microcontrolador PIC para la adquisición de datos de aceleración. El Slave a pedido del Master puede modificar los parámetros de rango de aceleracion, número de datos requeridos y cuando iniciar o detener la lectura de datos de aceleración. El Slave también envía al Master datos de voltaje de la bateria que alimenta a la tarjeta Slave. El envío de esta data se da mediante comunicación inalámbrica Bluetooth al Gateway para que este valide y envié la data al Master.

4.2. Análisis de parámetros y selección de equipos

4.2.1. Análisis de la frecuencia de muestreo

Como ya definimos la forma de operación de nuestro sistema de comunicación Master – Gateway - Slave, el siguiente paso es realizar un análisis de los mínimos requerimientos que deben cumplir los equipos a seleccionar con el fin de desarrollar nuestro proyecto. Como necesitamos adquirir datos de aceleración, nuestro primer paso debe ser seleccionar el elemento principal o sea nuestro acelerómetro, por lo tanto debemos verificar las condiciones reales en las que trabajaremos con el fin de definir parámetros, como la frecuencia de muestreo, rango de aceleracion, sensibilidad, interfaz de comunicación, entre otros, tomando como referencia artículos que describen el comportamiento de equipos de bombeo reales.

En la industria petrolera un equipo de bombeo mecánico súper veloz realiza aproximadamente 11 strokes por minuto, y uno en operación normal varía entre los 4 a 7 stroke por minutos. Si partimos de estos datos considerando el caso extremo de un pozo súper veloz tendríamos los siguientes datos:

Fp = 11 strokes/min

Fp = 0.18 strokes/seg

Tp ≈ 5.45 seg/strokes

$$Tm = \frac{Tp}{100} \approx 54 \text{ ms}$$

Donde:

Fp: Frecuencia de operación del pozo

Tp: Periodo de operación del pozo

Tm: Periodo de muestreo.

De acuerdo al análisis realizado tenemos un periodo de muestreo igual a 54ms lo que equivale a una frecuencia de 19 Hz. Si queremos tener como mínimo 100 puntos por periodo, deberíamos seleccionar una frecuencia de muestreo igual o mayor a 19 Hz. Ahora debemos definir el rango de aceleración que debe tener nuestro acelerómetro para medir los mínimos cambios de aceleración.

4.2.2. Análisis del rango de aceleracion

Como los cambios de aceleración varían entre 1 a 2g, entonces nuestro acelerómetro como mínimo debería tener un rango de aceleracion igual o superior a 2g. En cuestiones de lo que respecta a la temperatura debe estar en un rango de trabajo aproximado de 25 a 40 grados °C. Deberíamos optar por elegir un sensor del tipo discreto que maneje una interfaz de comunicación como RS232, I^2C o SPI en vez de un sensor de salida analógica con el fin de evitar hacer una conversión de los valores de voltaje a un valor numérico, lo cual como sabemos introduce errores de cuantizacion producidos por la aproximación o redondeo del proceso de conversión analógico-digital.

4.3. Acelerómetro ADXL345

Una vez analizadas las características que debe tener nuestro sensor, elegimos el acelerómetro ADXL345 el cual cumple con los mínimos requerimientos para nuestro sistema. Este es un acelerómetro MEMS digital de tres ejes fabricado por Analog Devices y entre sus características principales están las siguientes:

- Consumo de potencia ultra bajo, 145μA en modo de medición a 1.6kHz de ancho de banda y 0.1μA en modo espera a un voltaje de alimentación de 2.5V.
- Resolución seleccionable, por defecto de 10 bits, pero la resolución incrementa con la gama en "g" hasta un valor máximo de 13 bits a ±16g (manteniendo un factor de escala de 4 mg/LSB en todos los rangos de gravedad).
- Incorpora un sistema integrado FIFO de 32 niveles.
- Rango de temperatura: −40°C a 85°C
- Rango de alimentación: 2.5V y 3.6V.

- Interfaz de comunicación digital: I^2C o SPI (3 o 4 hilos).
- Medida de rangos: ±2g, ±4g, ±8g, ±16g
- Velocidad de salida de data: 6.25Hz, 12.5Hz, 25Hz, 50Hz, 100Hz, 200Hz, 400Hz, 800Hz, 1.6kHz, 3.2kHz.
- El consumo de potencia se ajusta al ancho de banda seleccionado.
- Usado en aplicaciones para medir impacto, en sistemas de navegación, así como aceleración estática de gravedad y aceleración dinámica del resultado de un movimiento o choque.

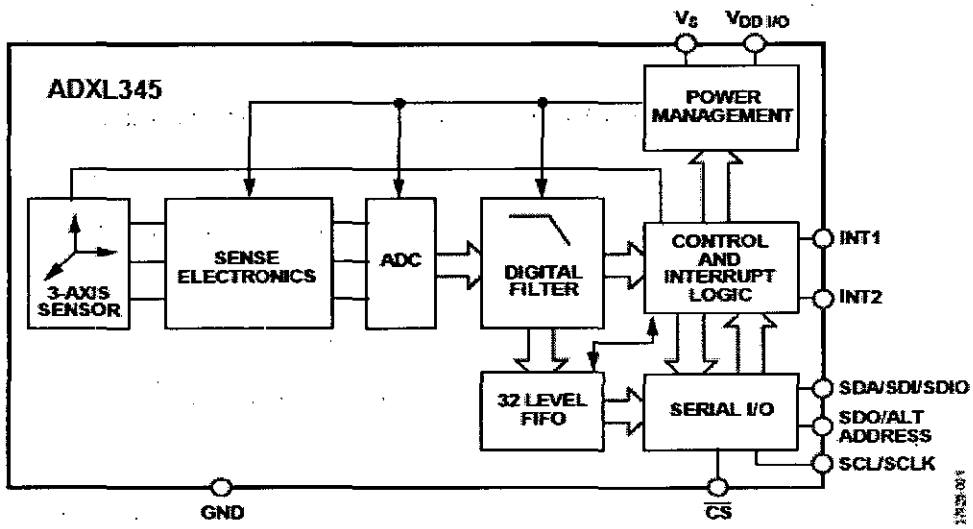


Figura 4. 4 Diagrama de bloques del ADXL345

4.4. PIC 18F2550

Una vez seleccionado nuestro acelerómetro el siguiente paso es seleccionar el microcontrolador mediante el cual realizaremos la adquisición de la data, envió de la misma y comunicación con la PC, por lo tanto nuestro microcontrolador debe contar como mínimo con las siguientes características:

- Periférico de comunicación USB
- Puerto de comunicación RS232
- Interfaz de comunicación I^2C/SPI
- Módulo ADC

El microcontrolador elegido es el PIC 18F2550 de microchip, el cual es un pic de gama media que posee los periféricos necesarios para poder realizar nuestra comunicación, como USB, UART, I^2C entre otras características importantes que se detallan a continuación:

- Modulo USB 2.0, en modo Low Speed (1.5Mbs/s) y Full Speed (12Mbs/s), chip regulador interno y chip Transceiver.
- Frecuencia de trabajo hasta 48MHz
- Transferencias de Control, Interrupt, Isochronous y Bulk Transfers.
- Memoria Flash: 32Kbytes
- Memoria EEPROM: 256 bytes
- Memoria SRAM: 2Kbytes
- modos de cristal incluyendo el PLL de gran precisión para USB
- 8 fuentes internas de reloj desde 31kHz hasta 8MHz
- Rango de operación de voltaje: 2.0v A 5.5V
- Consumo de corriente: 25mA
- Rango de temperatura: −40°C a 85°C
- módulos Timers: Timer 0, Timer 1, Timer 2, Timer 3.
- 2 módulos CCP (Captura/Comparación/PWM)
- 2 comparadores analógicos.
- Módulo USART, I^2C y SPI
- Convertidor A/D de 13 canales con una resolución de 10 bits.
- 24 pines de I/O
- 3 interrupciones externas

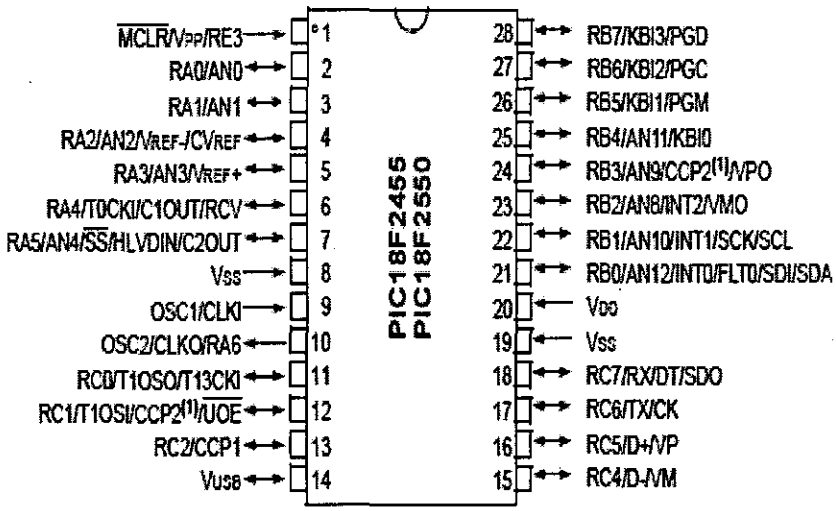


Figura 4. 5 Microcontrolador PIC 18f2550

4.4.1. Periférico USB

La familia PIC18FX455/X550 contiene un módulo USB full speed o low speed compatible con USB Serial Interface Engine (SIE) que permite la comunicación entre el Microcontrolador PIC y el USB Host.

El SIE puede interactuar directamente en el USB utilizando un "internal Transceiver" o puede ser conectado a través de un "external Transceiver". Un regulador interno de 3.3V puede ser activado (VREGEN – Enable), se requiere de un capacitor externo de 220nF (±20%) para estabilidad en VUSB. Cuenta también con un USB RAM de 1 Kbyte, donde almacenará datos enviados o recibidos y del buffer de control de los Endpoint.

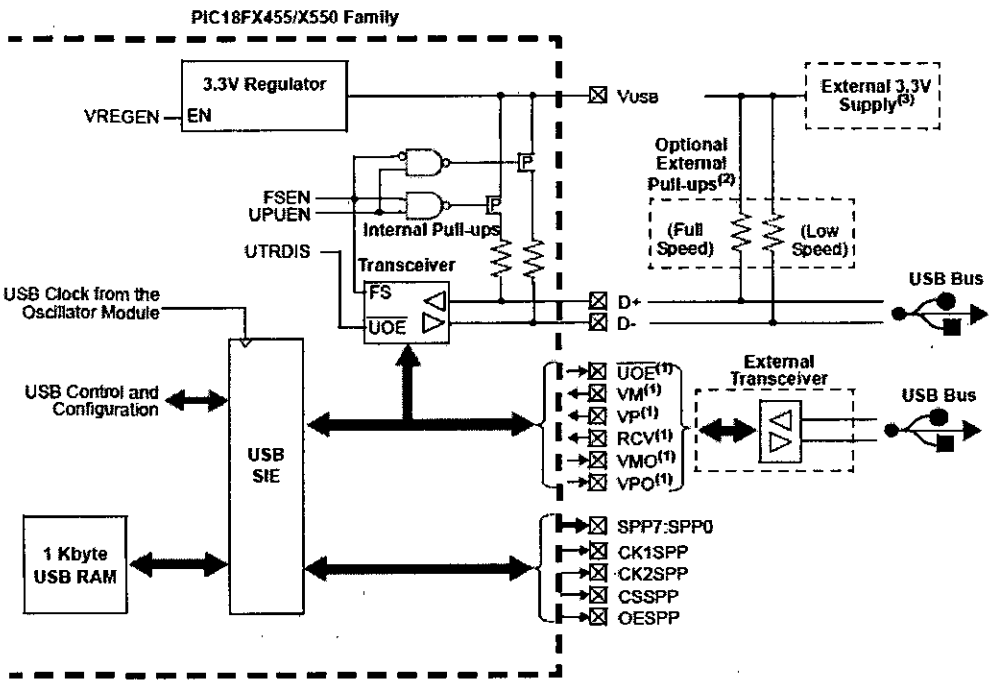


Figura 4. 6 Periférico USB del PIC18F2550

4.4.2. Configuración del reloj para USB

En la familia PIC18F2455/PIC18F2550/PIC18F4455/PIC18F4550 el oscilador primario forma parte del módulo USB y no se puede asociar a ninguna otra fuente de reloj. El módulo USB tiene que ser controlado por la fuente primaria y los otros dispositivos o el microcontrolador se pueden controlar por fuentes secundarias u osciladores internos como en el resto de microcontroladores PIC.

Al utilizar el USB necesitamos un reloj interno de 6MHz ó 48MHz, ya sea para trabajar al Low Speed o Full Speed, el resto del sistema puede funcionar con cualquier otro oscilador. En nuestra tarjeta ADQ para el Gateway utilizamos un cristal de 20MHz con sus respectivos condensadores de 15pF, y modificando los PLL y CPUDIV para obtener una frecuencia en el microcontrolador de 48MHz de frecuencia. En la figura 4.7 se muestra las opciones de configuración del oscilador.

Input Oscillator Frequency	PLL Divisor (PLLDIV2:PLLDIV0)	Clock Mode (FOSC3:FOSC0)	MCU Clock Divisor (CPUDIV1:CPUDIV0)	Microcontroller Clock Frequency
20 MHz	÷5 (100)	HS, EC, ECIO	None (00)	20 MHz
			÷2 (01)	10 MHz
			÷3 (10)	6.67 MHz
			÷4 (11)	5 MHz
		HSPLL, ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	24 MHz
			÷6 (11)	16 MHz
16 MHz	÷4 (011)	HS, EC, ECIO	None (00)	16 MHz
			÷2 (01)	8 MHz
			÷3 (10)	5.33 MHz
			÷4 (11)	4 MHz
		HSPLL, ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	24 MHz
			÷6 (11)	16 MHz
12 MHz	÷3 (010)	HS, EC, ECIO	None (00)	12 MHz
			÷2 (01)	6 MHz
			÷3 (10)	4 MHz
			÷4 (11)	3 MHz
		HSPLL, ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	24 MHz
			÷6 (11)	16 MHz
8 MHz	÷2 (001)	HS, EC, ECIO	None (00)	8 MHz
			÷2 (01)	4 MHz
			÷3 (10)	2.67 MHz
			÷4 (11)	2 MHz
		HSPLL, ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	24 MHz
			÷6 (11)	16 MHz
4 MHz	÷1 (000)	XT, HS, EC, ECIO	None (00)	4 MHz
			÷2 (01)	2 MHz
			÷3 (10)	1.33 MHz
			÷4 (11)	1 MHz
		HSPLL, ECPLL, XTPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	24 MHz
			÷6 (11)	16 MHz

Figura 4. 7 Opciones de configuración del oscilador para el modulo USB

4.4.3. CONVERTIDOR ANALÓGICO DIGITAL

Los microcontroladores PIC pueden incorporar un módulo de conversión de señal analógica a señal digital. El PIC 18f2550 cuenta con módulo ADC que consta de 13 canales a una resolución de 10 bits los cuales se multiplexan para permitir la lectura de varios canales. Además nos da la opción de elegir el nivel de referencia con respecto a GND y VDD o también con respecto a dos niveles de referencia diferente. En la figura 4.8 se muestra el diagrama de bloques del convertidor A/D.

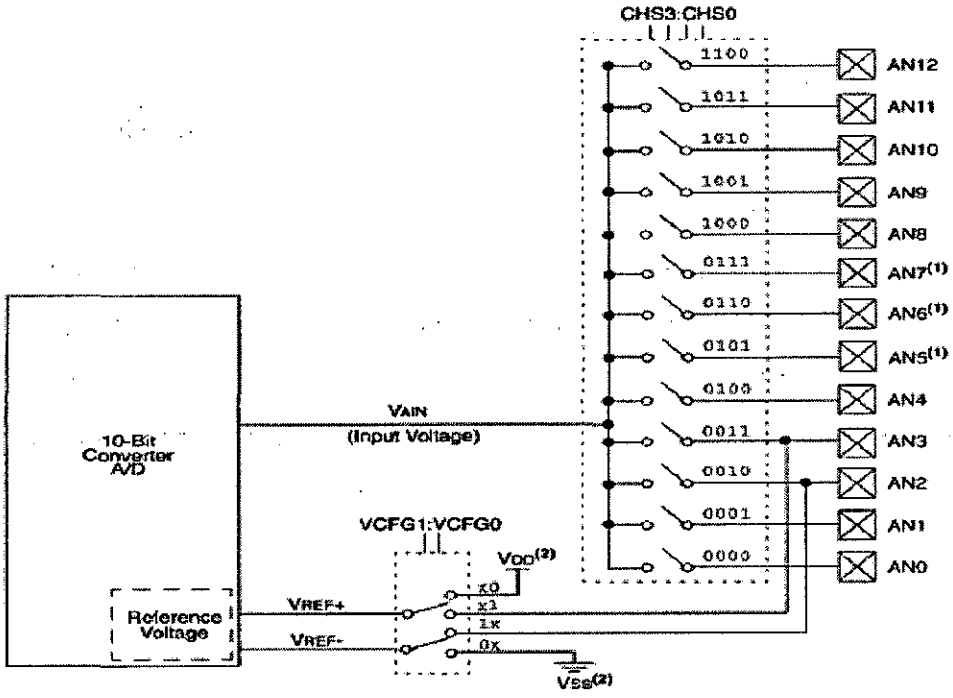


Figura 4. 8 Diagrama de bloques del convertidor A/D

El modulo A/D que utiliza Microchip hacen un muestreo y retención (sample & hold) con un condensador y después utiliza el módulo de conversión utilizando aproximaciones sucesivas, como se muestra en la figura 4.9.

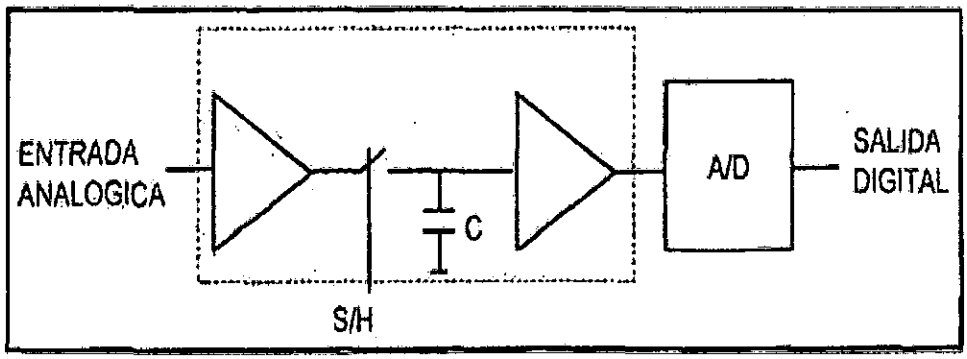


Figura 4. 9 Fases de la conversión Analógica/Digital

Durante las fase de muestro el interruptor se cierra y el condensador se carga a tensión de entrada (el tiempo que el interruptor permanece cerrado es fundamental para la correcta carga del condensador). Una vez abierto el interruptor, el condensador mantendrá (teóricamente) la tensión de entrada mientras el módulo A/D realiza la conversión.

4.5. BLE 4.0

El modulo Bluetooth seleccionado para la comunicación entre el Slave y el Gateway es el perteneciente a la tecnología 4.0 o conocido como BLE. La elección de utilizar este tipo de Bluetooth es porque debido a que la alimentación del Slave se realiza mediante una batería y no mediante un sistema de alimentación fotovoltaica, el consumo de energía es un aspecto importante a tener en cuenta. Este BLE está orientado a aplicaciones donde se requiera un bajo consumo de energía ya que consume 3 a 5 veces menos que un Bluetooth clásico. Entre las principales características de este módulo están:

- Ultra bajo consumo: 60μA –1.5mA en modo Sleep y 8.5mA en modo activo.
- Voltaje de entrada: 3.6V-6VDC
- Frecuencia de trabajo: 2.4GHz ISM band
- Potencia RF: -23dbm, -6dbm, 0dbm, 6dbm modificadas mediante comandos AT. Por defecto viene a 0dbm
- Antena incluida
- Baudios: 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400. Por defecto viene a 9600 baudios.

- Puede ser usado como Maestro o Esclavo
- Cobertura hasta 20 metros campo abierto.
- Temperatura de trabajo: −5°C a 65°C
- Configurable mediante comandos AT

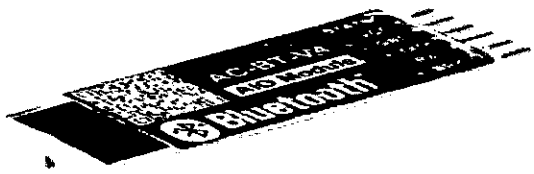


Figura 4. 10 Modulo Bluetooth 4.0

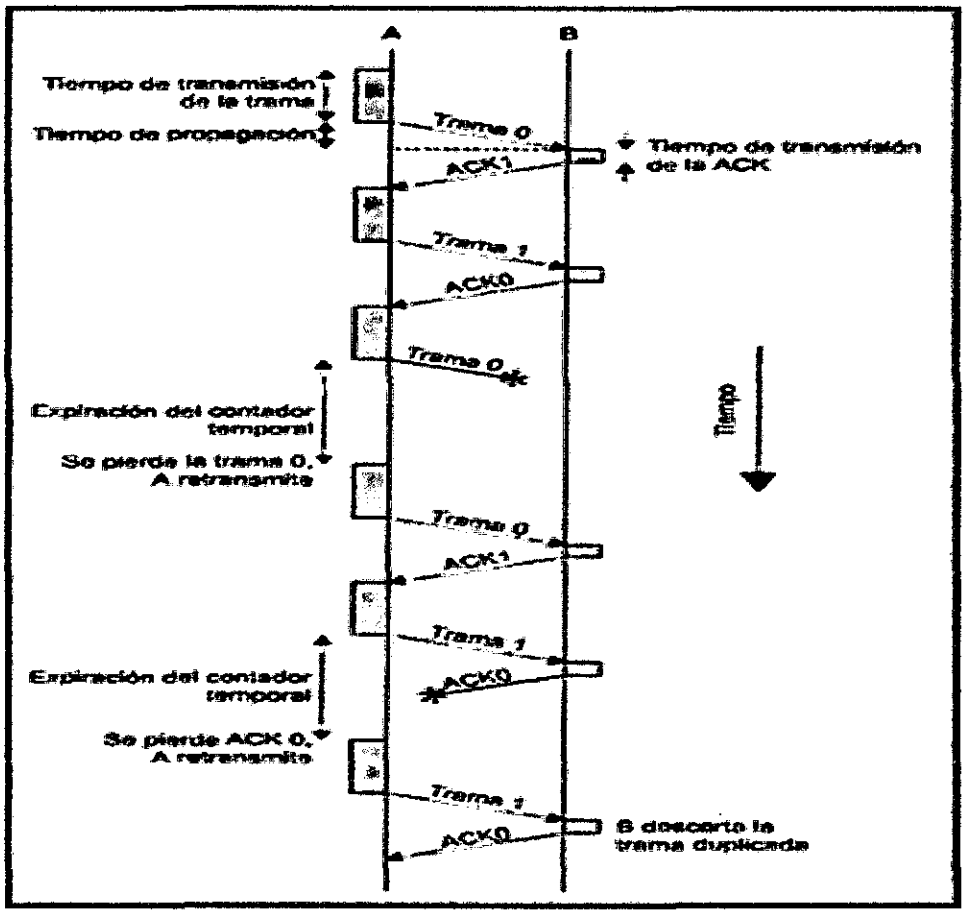


Figura 5. 1 Secuencia de envío y recepción en un protocolo Stop and Wait

5.2. Problemas a considerar en el diseño del protocolo

Durante él envió y recepción de paquetes pueden darse los siguientes escenarios que afectan la comunicación y que se detallan a continuación:

- Supongamos que el emisor envía un paquete al receptor, este recibe el paquete, verifica que es válido y envía el ACK de vuelta, pero en el retorno ***la trama de confirmación de recepción o de datos se daña o se pierde por completo*** o en el peor de los casos nunca llega. El emisor al no recibir el ACK, reenvía la trama nuevamente. Si no existiera un tiempo de espera para recibir el paquete de confirmación, este se quedaría esperando infinitamente a que llegue la confirmación del paquete lo cual es totalmente inadmisible en cualquier protocolo.

- Si se reenvía un paquete debido a algún error, el receptor tendrá un ***duplicado de trama*** y no sabe si la segunda es una trama duplicada o si es la siguiente trama de la secuencia que enviara, que en realidad contiene datos idénticos a la primera. Si consideramos una comunicación en donde ambos asumen la función de emisor y receptor, se origina una ***pérdida de secuencia y un desorden*** en los paquetes.
- Otro problema que surge es la ***latencia*** que está presente en el medio de transmisión, si es mayor que el tiempo de espera del emisor, se termina incluso antes de que la trama llegue al receptor. En este caso, el emisor reenvia la trama. Eventualmente el receptor tiene un duplicado de trama y envía un ACK por cada paquete. Entonces, el emisor que esta a la espera de un solo ACK, recibe dos ACK's que puede causar problemas si el emisor asume que en el segundo ACK es para la siguiente trama en la secuencia.

5.3. Definición del protocolo de comunicación

Considerando los posibles problemas que se pueden presentar en él envió de paquetes, podemos definir un protocolo que considere todos los casos anteriores, que sea lo suficientemente compacto y que nos garantice una comunicación confiable. En esta investigación se ha diseñado un sistema robusto, capaz de detectar y recuperarse de errores antes mencionados.

El formato del paquete de comunicación consta de:

- El paquete tiene una cabecera (*Header*) y un final de cabecera (*End of Header*) definidos según la tabla ASCII como STX (*Start of text*) y ETX (*End of text*).
- Utilizamos una suma de comprobación o también llamado *checksum,* con el cual verificamos que el paquete que recibimos llega libre de errores y no ha sido modificado en el transcurso que viaje por el canal.
- Se envía la confirmación (ACK) o confirmación negativa (NACK) del paquete recibido dentro del mismo paquete y no por separado.

- Se incluye un secuenciador de paquete de un byte (0/1) para poder distinguir paquetes retransmitidos y mantener el sincronismo de envío y recepción de paquetes.
- Se define un tiempo maximo de espera de ACK controlado por el Master, para que este no se quede esperando por un paquete de respuesta que no llega por algún error de comunicacion.
- Tanto Master; Slave y Gateway, realizan una validación del paquete. Esta validación consiste en lo siguiente:
    1. Que el paquete comience con STX y termine con ETX.
    2. Que el checksum del receptor sea el mismo que el del emisor.

Si se cumplen estas dos condiciones el paquete es admitido y luego se procede a verificar si se recibe un ACK, si la secuencia de paquete es la correcta y finalmente se envía un paquete de respuesta de acuerdo al tipo de paquete que solicita el Master.

5.4. Formato de paquete de comunicación Master

El formato del paquete de comunicación utilizado por el Master se muestra en la figura 5.2.

STX	ACK	SEC	CMD	DATA	CHSM	ETX

Figura 5. 2 Formato de paquete de envío Master

Son 7 bytes enviados por el Master al Gateway, los cuales están en código ASCII y cada campo del paquete se describe a continuación.

- **STX:** Es la cabecera del paquete el cual según la tabla ASCII tiene un valor decimal de 2.
- **SEC:** Es el secuenciador de paquete de un byte, por lo que puede tomar un valor de 0 o 1.
- **ACK/NACK:** Confirmación de recepción positiva o negativa del paquete con un valor decimal de 6 para el ACK y de 21 para el NACK.

- **CMD:** Es el campo que define el tipo de comando del Master al Slave. Los comandos son los siguientes:
    - "A" : Solicitar datos de aceleración
    - "G" : Establecer rango de aceleración en el acelerometro
    - "M" : Colocar acelerómetro en modo medición
    - "L" : Solicitar voltaje de batería
    - "R" : Establecer frecuencia de muestreo en el acelerometro
    - "T" : Establecer conexión Gateway-Slave
    - "S" : Colocar acelerómetro en modo standby
    - "V" : Desconectar Gateway-Slave

  Nota: Los comando "T" y "V" no son recepcionados por el Slave, solo por el Gateway y este se encarga de enviar los comandos AT a los cuales solo el modulo BLE Slave responde automáticamente.

- **DATA:** Solo se utiliza para seleccionar el rango de aceleracion y frecuencia de muestreo. Los valores de data utilizados para la selección son:

  Rango de aceleracion:

    - "1" : para seleccionar un rango de aceleracion de +/- 2g
    - "2" : para seleccionar un rango de aceleracion de +/- 4g
    - "3" : para seleccionar un rango de aceleracion de +/- 8g
    - "4": para seleccionar un rango de aceleracion de +/- 16g

  Frecuencia de Muestreo:

    - "1" : para seleccionar una taza de muestreo de 12.5Hz
    - "2" : para seleccionar una taza de muestreo de 25Hz
    - "3" : para seleccionar una taza de muestreo de 50Hz
    - "4" : para seleccionar una taza de muestreo de 100Hz

- **CHSM:** Es el campo de suma de verificación de paquete. Se realiza la suma de todo el paquete, desde el STX hasta el ETX y solo es un byte.
- **ETX:** Es el terminador de paquete y tiene un valor decimal de 3.

5.4.1. Funciones del Master

El master utiliza tres funciones realizadas en Matlab para el manejo de paquetes; una llamada ***SendtoGateway*** que se encarga de organizar y enviar el paquete de comunicacion al Gateway de acuerdo al formato definido anteriormente. La función ***ReadfromGateway*** que es la encargada de realizar la validación definida en el apartado 5.3, de los paquetes del voltaje de batería, conexión y desconexión del Gateway-Slave, modo standby e inicio de la medición de datos de aceleración, rango de aceleración y tasa de muestreo. La última función llamada ***CheckData*** se encarga de la validación del paquete de aceleración y de decidir que paquete de respuesta enviar si se producen condiciones de buffer full, buffer vacío, si se produce un timeout, si llega un paquete invalido o si se recibe un paquete diferente al de aceleración.

5.5. Formato de paquete de comunicación Slave

El formato del paquete de comunicación de Slave a Master se muestra en la figura 5.3:

STX	ACK	SEC_PKT	CMD	SEC_DAT1	...	SEC_DAT5	DAT1	...	DAT30	CHSM	ETX

Figura 5. 3 Formato de paquete de envío Slave

Son 45 bytes enviados por el Slave al Gateway, los cuales son enviados como caracteres ASCII y cada campo del paquete se describe a continuación.

- **STX:** Es la cabecera del paquete el cual según la tabla ASCII tiene un valor decimal de 2.
- **SEC_PKT**: Es el secuenciador de paquete de un byte, por lo que puede tomar un valor de 0 o 1.
- **ACK:** Confirmación de recepción positiva o negativa del paquete con un valor decimal de 6 para el ACK y de 21 para el NACK.
- **CMD:** Es el campo que define el tipo de comando de repuesta del Slave al Master. Los comandos son los siguientes:
  - ✓ "A" : Datos de aceleración
  - ✓ "E" : Buffer de datos de aceleración vacío

- ✓ "F" : Buffer de datos de aceleración lleno
- ✓ "G" : Confirmación de selección de rango de aceleración
- ✓ "M" : Confirmación de acelerómetro en modo medición
- ✓ "L" : Voltaje de batería
- ✓ "N" : Recepción de paquete invalido
- ✓ "R" : confirmación de selección de tasa de muestreo
- ✓ "S" : Confirmación de acelerómetro en modo standby

- **SEC_DAT1 – SEC_DAT5:** Este campo de tamaño de 5 bytes indican la secuencia de datos de aceleración, donde cada digito se envía como un carácter ASCII.
- **DAT1 – DAT 30:** Es el campo de Payload o carga útil del paquete el cual tiene un tamaño de 30 bytes, debido a que enviamos cinco datos del eje de aceleración seleccionado, los cuales tienen formato complemento a dos, con un tamaño de 2 bytes con signo. La razón por la cual enviamos cinco datos de aceleración en vez de uno es debido al delay (retardo) que se presenta en la trasmisión y recepción vía Bluetooth; el cual es de aproximadamente 40 ms, por lo tanto tendríamos un retardo aproximado de 80 ms, independientemente de la velocidad en que se transmita el paquete. Entonces muestreando a 25Hz; nuestro acelerómetro tiene un nuevo dato cada 40 ms y se corre el riesgo de que se produzca un overflow en el buffer circular por lo cual se decidió enviar cinco datos en cada solicitud de data de aceleración. Como la medida de cada eje representa cada digito por su carácter ASCII tendríamos un total de 30 bytes de acuerdo al tipo de paquete que se solicita, el campo de datos útiles puede variar, pero el tamaño queda fijo.
- **CHSM:** Es el campo de suma de verificación de paquete. Se realiza la suma de todo el paquete, desde el STX hasta el ETX y tiene un tamaño de un byte.
- **ETX:** Es el terminador de paquete y tiene un valor decimal de 3.

5.5.1. Descripcion del buffer circular de data usado en el Slave

Para el almacenamiento de data de aceleración en el Slave, se optó por utilizar un buffer del tipo circular. Este buffer es una estructura de datos que utiliza un buffer único y que adopta su nombre por la forma en como se ponen o se sacan sus elementos, en otras palabras un buffer del tipo circular es bien visto como un buffer FIFO (primero en entrar es el primero en salir). Normalmente cuando la frecuencia de transferencia de datos es distinta a la de procesado, dependiendo de las limitaciones del sistema, las diferencias de tiempos que existen en la transmisión son generalmente ajustadas mediante la implementación de un algoritmo tipo cola (o estructura FIFO) en memoria, para así escribir datos en la cola a una frecuencia y leerlos a otra. En nuestro caso el proceso de comunicación presenta una situación similar, existe un retardo en la línea de transmision propio del protocolo, lo cual origina que tendríamos datos nuevos de aceleración a una velocidad mayor de la que se envían para ser procesados. Por lo tanto la opción de un utilizar un buffer circular es propicia para nuestro proceso de comunicación.

El modo de operación del buffer consta básicamente de dos punteros, uno de lectura y otro de escritura los cuales tienen un avance incremental y cíclico, para acceder a los elementos del buffer. Sin embargo hay que tener cuidado en realizar un buen manejo del buffer circular, para evitar las condiciones de ***overflow***; el cual se produce cuando el puntero de escritura recorrió el buffer y alcanza al puntero de lectura ocasionando que el buffer se llene, lo cual puede ser ocasionado debido a que no se leen datos del buffer o la lectura es más lenta que la escritura, y la condición de ***underflow*** que se produce cuando el puntero de lectura recorre el buffer y alcanza al puntero de escritura lo cual indica que el buffer se encuentra vacío y por lo tanto no hay datos disponibles en el para ser leídos. Para nuestra investigación definimos una estructura de datos para almacenar lo datos de aceleración de los 3 ejes del acelerómetro y la secuencia de datos, además de un tamaño del buffer circular de 20 elementos.

5.6. Funciones del Gateway

El Gateway es el encargado de enlazar dos protocolos. Por un lado el protocolo Bluetooth que le permite comunicarse con el Slave y por otro lado el protocolo USB con el cual se comunica con el Master. Ademas realiza la validación de paquetes mencionada en el apartado 5.3 y que recibe tanto del Master como el Slave, los retransmite siempre y cuando sea un paquete valido, por lo cual adopta el mismo formato del paquete de comunicacion que recibe, así que este no tiene un formato de paquete definido. Lo que sí es importante tener en cuenta es que el paquete de comunicación que envía el Gateway al Master necesariamente es del tipo entero de 1 byte sin signo (0-255), debido que a así lo indica la librería que comunica por USB al Master con el Gateway y con un tamaño máximo de 64 bytes tanto para el Endpoint de transmisión como el de recepción.

5.7. Selección de la velocidad y timeout para la transmisión de paquetes

Para definir el tiempo de espera desde que enviamos el paquete desde el master hasta que el Slave nos devuelva una respuesta y llegue a la PC, debemos calcular el tiempo que demora en viajar el paquete hasta el Slave ida y vuelta y además tener en consideración el retardo producido por la transmisión del paquete mediante Bluetooth que es de 80ms; 40ms para él envió y 40ms para la recepción del paquete, y un tiempo de estabilización del acelerómetro, con la finalidad de evitar un timeout. Entonces entendido esto, comencemos primero por definir la velocidad de transmisión a la cual debemos enviar nuestro paquete de comunicacion.

Al definir la velocidad de transmisión, nuestro objetivo es transmitir los datos de forma que en nuestro buffer circular no se produzca una condición de ***overflow***; debido a que estamos leyendo más lento de lo que se escribe, por lo tanto, debemos mantener una diferencia entre el puntero de escritura (***iw***) con el puntero de lectura (***ir***) debe ser diferente de cero. Si tenemos el tiempo en que nuestro acelerómetro tiene un nuevo dato listo para ser leído y almacenado en nuestro buffer, que es de 40 ms y el tamaño máximo del paquete del Slave es de 45 bytes, partimos seleccionando una velocidad de transmisión de 9600 baudios para analizar si es la correcta.

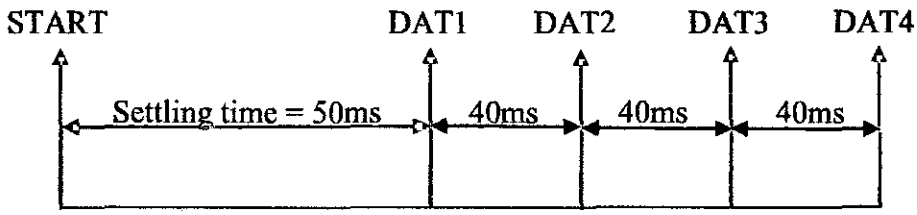


Figura 5. 4 Secuencia de inicio y datos en el acelerómetro ADXL345

V_{tx} = 9600 bits/s = 960 bytes/s

$T_{tx} \approx$ 1,04 ms/byte.

Entonces si transmitimos 52 bytes (45 de Slave-Gateway y 7 de Gateway-Slave) tendríamos un tiempo de tiempo de transmisión de aproximadamente 53ms.

Por lo tanto el tiempo de espera está dado por:

time_wait = time_tx + time_bt + settling_time + time_validacion_data

Para una V_{tx} = 9600 baudios tenemos:

Time_wait = 53 ms + 80 ms + 50 ms + 5 ms

Time_wait ≈ 200 ms.

El tiempo de establecimiento; denotado por settling time, es el tiempo que necesita el acelerometro cuando cambia del modo standby al modo medición, y varía de acuerdo al data rate que hemos seleccionado. Según la hoja de datos del acelerómetro se debe esperar un tiempo $\tau = 1.1 + 1/(\text{data rate})$.

Entonces como vemos si seleccionamos una valeocidad de transmisión de 9600 baudios, cuando enviamos el paquete de inicio o start hasta que el master reciba el paquete de respuesta debemos esperar un poco más de los 200 ms para no estar al límite. Por lo tanto cuando enviamos el paquete de aceleración, nuestro buffer circular ya tendría 5 datos escritos, sumado al retardo del Bluetooth que es de aproximadamente 160 ms que implica 4 datos mas, corremos el riesgo de que este se llene a medida que se recorre el buffer o por causa de errores en el proceso de

comunicación que originen una retransmisión de paquetes. Se podría trasmitir a eta velocidad pero a costa de eso el tamaño de buffer circular deberia ser mayor. Por lo tanto debemos seleccionar una velocidad de transmisión mayor.

Si elegimos una V_{tx} = 38400 baudios tenemos:

V_{tx} = 38400 bits/s = 3840 bytes/s

$T_{tx} \approx 0.25$ ms/byte, entonces si transmitimos 52 bytes, tendríamos un tiempo de tiempo de transmisión de aproximadamente 13 ms.

Por lo tanto el tiempo de espera está dado por:

time_wait = time_tx + time_bt + settling time + time_validacion_data

Para una V_{tx} = 38400 baudios tenemos:

Time_wait = 13 ms + 80 ms + 50 ms + 5 ms

Time_wait ≈ 150 ms.

Ahora con una velocidad de 38400 baudios, solo tendríamos aproximadamente tres datos de aceleración escritos en el buffer, más 4 datos ya escritos debido al retardo por Bluetooth, tendríamos 7 datos en el buffer y con lo cual la diferencia entre el puntero de escritura y lectura seria de 2 con lo cual nuestro buffer circular estaría menos expuesto a las condiciones de overflow ya que se asegura que siempre existirá como mínimo seis datos nuevos para leer. Por lo tanto nuestros modulo bluetooth y el puerto de comunicación de nuestro microcontrolador debe estar configurado para trabajar con una velocidad de transmisión de 38400 baudios.

Este registro nos indica que si colocamos algún bit a 1 se habilitan las funciones para generar la correspondiente interrupción, por el contrario un valor de 0 en algún bit impide que las funciones generen interrupciones. Los D7, D1 y D0 permiten sólo la salida de interrupción; las funciones están siempre habilitadas. Además se recomienda que las interrupciones sean configuradas antes de activar sus salidas. Entonces de acuerdo a esto el valor del registro INT_MAP debe ser 0X80.

5.8.2. Configuración de módulo BLE 4.0

Para configurar nuestro dispositivo Bluetooth utilizaremos el conocido Hyperterminal y un adaptador de USB a RS232 para poder enviar y recibir los comandos AT, mediante los cuales podemos configurar el modulo, de acuerdo a la hoja de datos que nos provee el fabricante.

Para configurarlo debemos iniciar el hyperteminal y seguir lo siguientes pasos que se describen a continuacion:

1) Hacer click en el boton inicio o el simbolo de windows en la parte inferior izquierda, buscamos el programa hyperterminal e iniciamos. Una vez iniciado el programa le damos un nombre cualquiera y le damos OK.

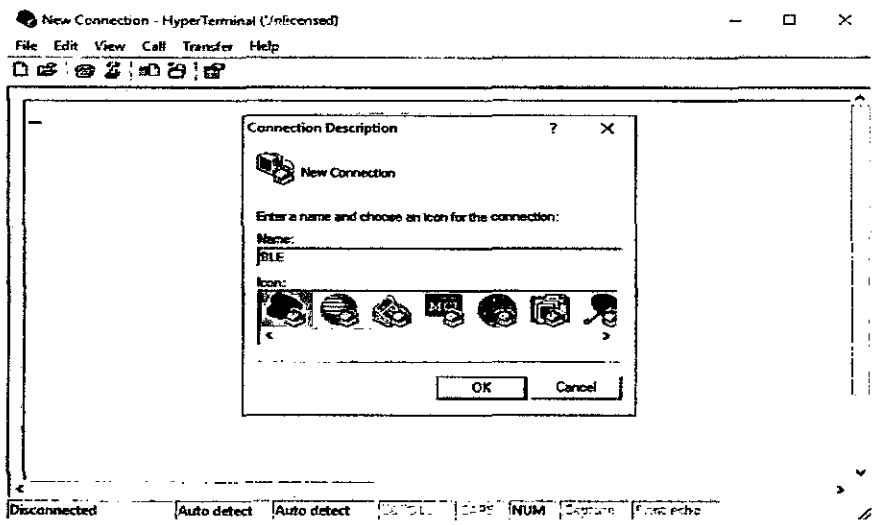


Figura 5. 12Inicialización del programa Hyperterminal

2) El siguiente paso es seleccionar el puerto de comunicación que el sistema operativo le asigno a nuestro adaptador USB-RS232. En nuestro caso es el COM8. Lo seleccionamos y damos OK como se muestra en la siguiente figura. Si no sabemos que COM le asigno el sistema operativo vamos al icono de inicio y escribimos "administrador de dispositivos" y luego en la seccion PUERTOS COM verificamos el COM asignado, como se muestra en la figura 5.13.

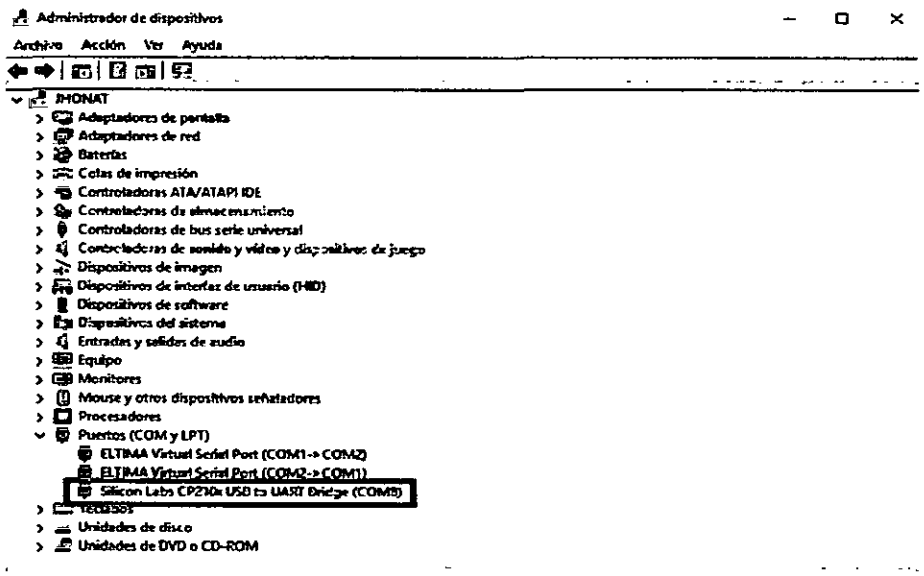


Figura 5. 13Verificación del puerto COM del adaptador USB-RS232

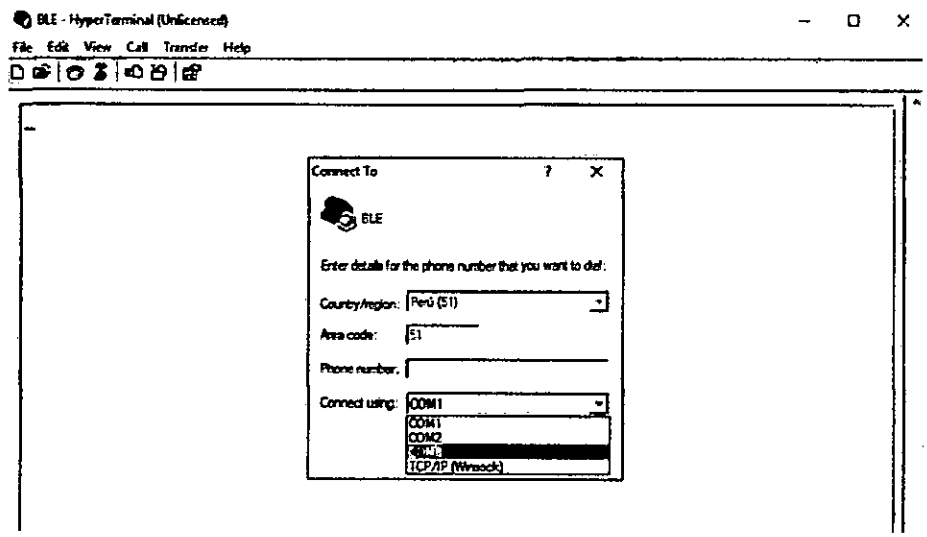


Figura 5. 14 Selección del puerto COM asignado al adaptador USB-RS232

3) Una vez seleccionado el puerto, debemos configurar la velocidad de transmision a la misma velocidad con la cual viene de fabrica nuestro modulo bluetooth. Por defecto el modulo viene configurado a una velocidad de transmision de 9600 baudios y sin control de flujo, por lo tanto debemos seleccionar en nuestro hyperterminal los mismos parametros como se muestra en la siguiente figura.

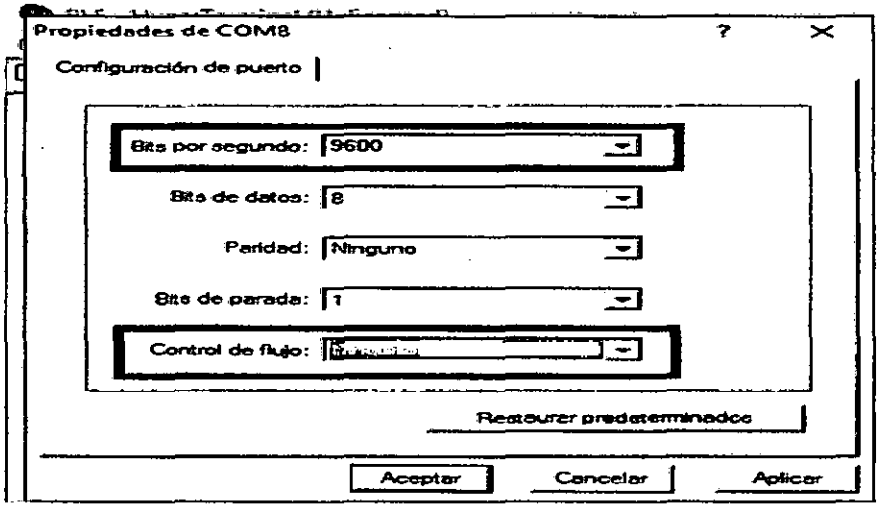


Figura 5. 15 Configuración de baudios y control de flujo en Hyperterminal

4) Ahora nos dirigimos al icono de propiedades el cual está situado es el último icono de la barra de herramientas o de lo contrario ir al menú File y seleccionar propiedades. Luego nos dirigimos a la pestaña **"Settings"** y seleccionamos al final el botón **"ASCII SETUP"**. Una vez aquí seleccionamos las dos primeras casillas, con la finalidad de hacer un eco y poder visualizar los comandos que escribimos en la pantalla de Hyperterminal además de enviar un fin de línea después de cada comando, como podemos visualizar en las figuras 5.16 y 5.17.

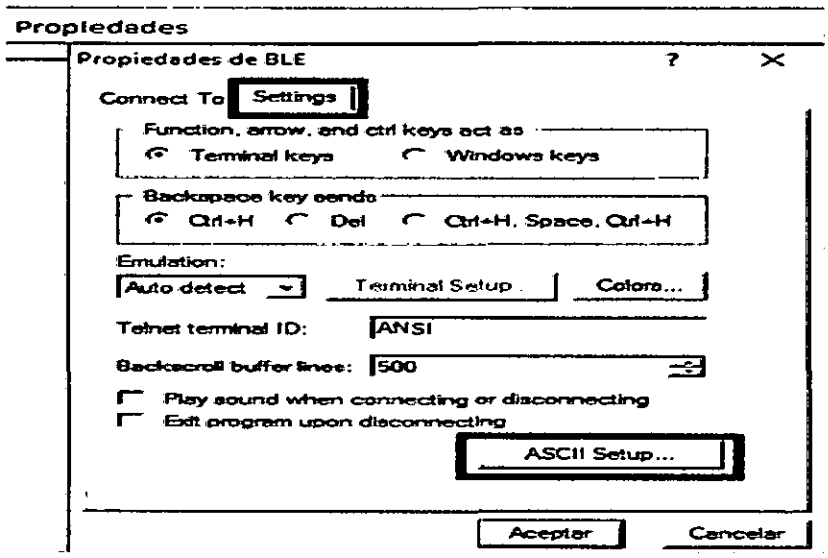


Figura 5. 16 Selección del menú de configuración ASCII

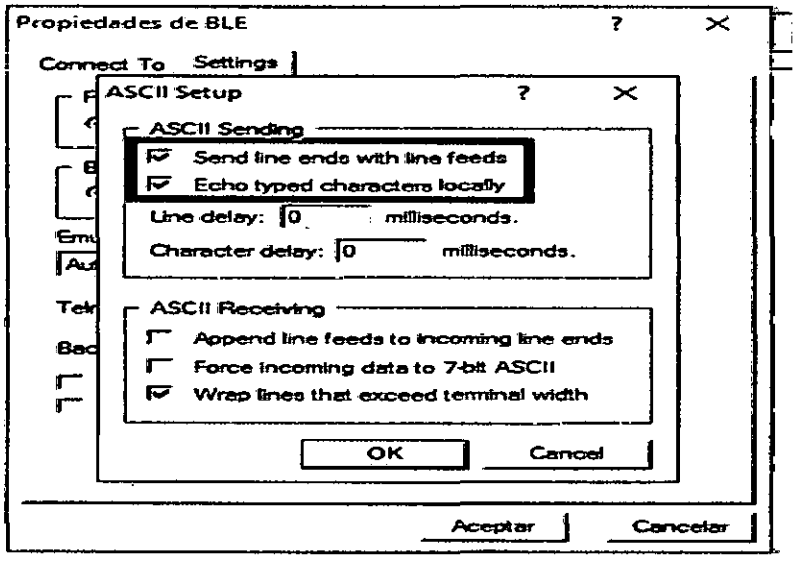


Figura 5. 17 Selección de eco y envió de carácter fin de linea.

5) Ahora estamos listos para empezar a configurar nuestro modulo Bluetooth, para lo cual necesitamos primero la lista de comandos AT que permiten configurar al modulo y segundo utilizar el block de notas para escribir los comandos AT. Usted se preguntara porque utilizar el block de notas si en Hyperterminal lo podrian hacer, pero la razon es la siguiente. La mayoria de modulos; por no decir todos, se configuran mediante comandos AT y utilizan la misma palabra como comando para realizar un test de comunicación. Cuando enviamos **"AT"** el modulo nos responde inmediatamente con un OK si es que la

- **AT+MODE2:** comando para configurar el modo de trabajo en modo remota, que incluye la configuracion del modulo mediante comandos AT y el posterior envio de estos una vez establecida la conexion.
- **AT+POWE3:** comando para configurar la potencia de trabajo del modulo a 6dbm.
- **AT+BAUD2:** comando para configurar la velocidad de transmision a 38400 baudios.

Para realizar la configuracion del modulo, previamente debe tener alimentacion en los pines Vcc y Gnd, ademas cruzar las lineas de comunicación del modulo con el adaptador, esto es Tx del adaptador a Rx del modulo y Rx del adaptador a Tx del modulo.

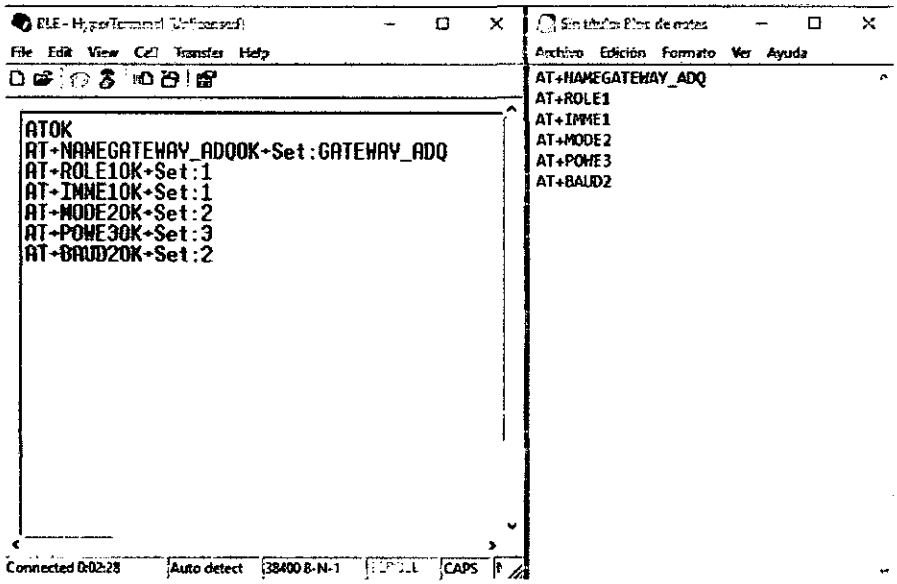


Figura 5. 18 Configuración del módulo Bluetooth como Master.

Una vez enviado el comando AT+BAUD2 que modifica nuestra nueva tasa de baudios a 38400, debemos nuevamente configurar nuestro Hyperterminal a 38400 baudios en caso queramos realizar otras configuraciones. Para esto hacemos click en el icono del telefono, luego click al icono de propiedades y finalmente click al boton Configure como se muestra en la figura 5.19.

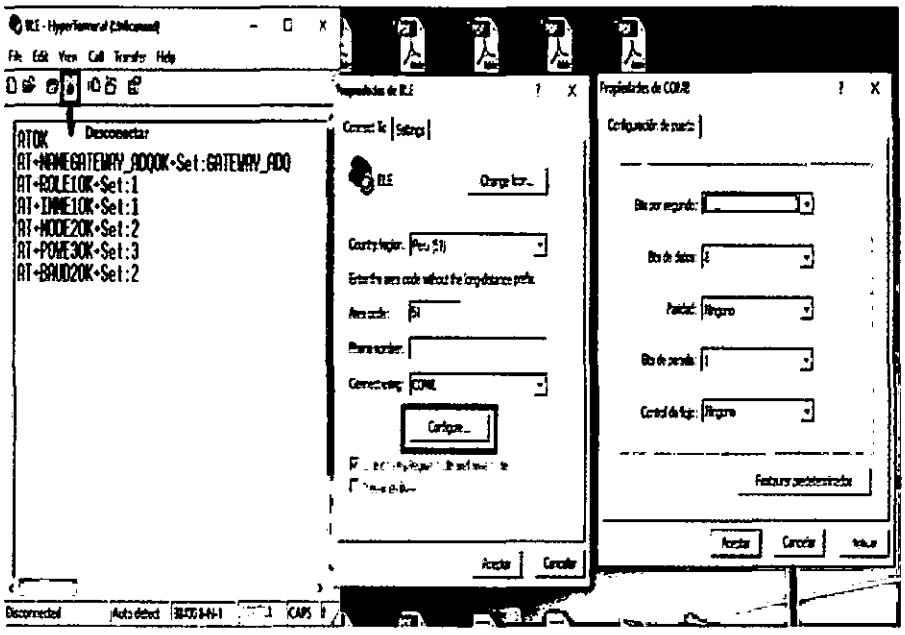


Figura 5. 19 Configuracion de la nueva tasa de baudios en Hyperterminal.

7) Para configurar el modulo que va conectado al acelerometro como Slave, usamos los mismos comandos pero con excepción del nombre y el tipo de dispositivo como bluetooth diferentes como se muestra a continuacion:

   - AT+NAMESLAVE_ADQ: comando para configurar el nombre. En nuestro caso decidimos llamarlo SLAVE_ADQ.
   - AT+ROLE0 : comando para configurar el modulo como Slave.
   - AT+MODE2 : comando para configurar el modo de trabajo en modo remota, que incluye la configuracion del modulo mediante comandos AT y el posterior envio de estos una vez establecida la conexion.
   - AT+POWE3 : comando para configurar la potencia de trabajo del modulo a 6dbm.

- AT+BAUD2 : comando para configurar la velocidad de transmision a 38400 baudios.

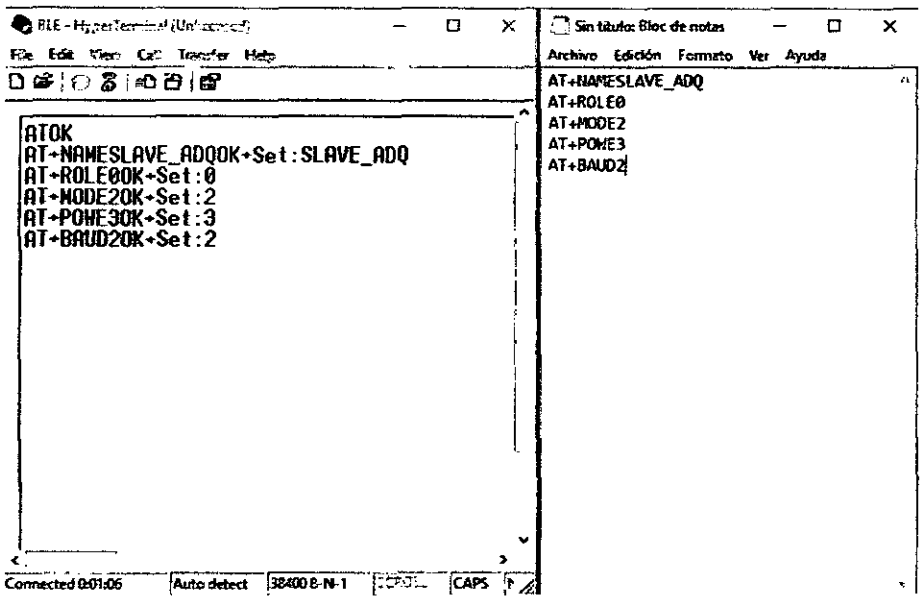


Figura 5. 20 Configuracion del modulo Bluetooth como Slave

5.8.3. Instalación del driver para comunicación USB

El driver que necesitamos para establecer la comunicación USB es proporcionado por Microchip en su página oficial www.microchip.com, en la sección *Application and markets //USB// MCHPFSUSB Framework// Software/Tools//* << Microchip Application Libraries v2010-02-09, o de lo contrario en el siguiente blog http://colab-matlab.blogspot.pe/2012/03/transferencia-de-datos-via-usb-con.html, en la sección final encontramos el enlace para la descarga del driver en format Zip.

Este driver contiene una lista de campos VID & PID mediante el cual el host realiza el proceso de enumeración del nuevo dispositivo que se ha conectado para posteriormente asignarle el driver correspondiente. Entonces al momento de desarrollar la programación del Gateway que es el microcontrolador que se comunica mediante USB al host; en este caso la PC,

1) Ejecute el driver descargado en el paso anterior como administrador, para esto damos clic derecho en el instalador, luego siga los pasos mencionados en el menú de instalación, e instale en la dirección que trae ya predeterminada.

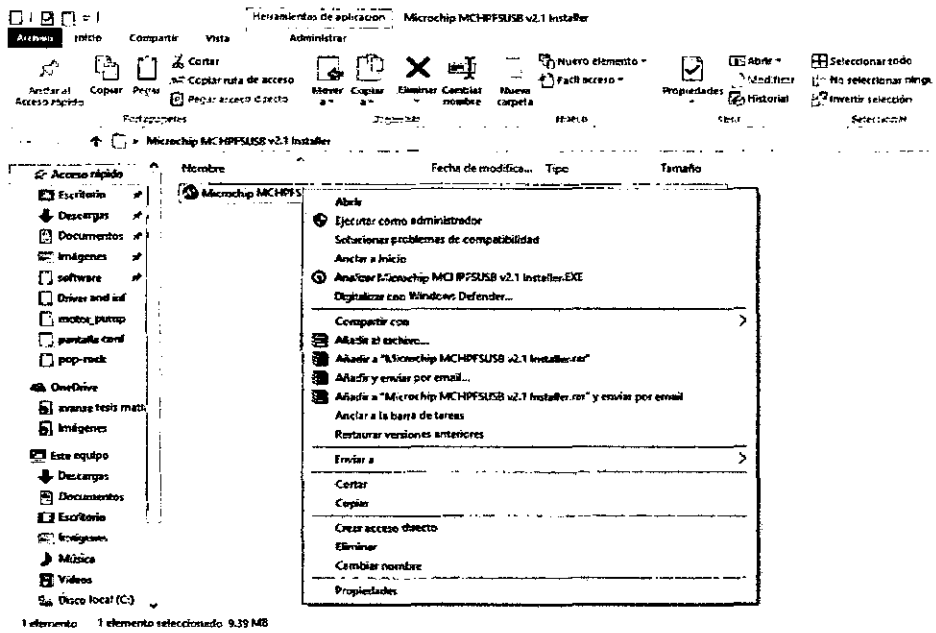


Figura 5. 21Iniciando el instalador Microchip MACHFUSB V2.1

2) Una vez terminado el proceso de instalación nos dirigimos al disco C y ubicamos la carpeta que tiene por nombre ***"MICROCHIP SOLUTIONS"***. En esta carpeta se encuentra ubicado nuestro driver en la siguiente ruta: *C:\MICROCHIP SOLUTIONS\USB TOOLS\MCHPUSB CUSTOM DRIVER\MCHPUSB DRIVER\ RELEASE*.
3) El siguiente paso es conectar nuestra tarjeta de ADQ para que pueda ser enumerado y poder cargar el driver del controlador, para poder establecer la comunicación. Nos dirigimos al administrador de dispositivos y podemos observar en la sección otros dispositivos que nuestra tarjeta no tiene un controlador instalado por lo tanto no es reconocido por la pc. Entonces lo que prosigue es dar clic derecho y seleccionamos la primera opción, "**actualizar software de controlador**" como se muestra en la siguiente figura.

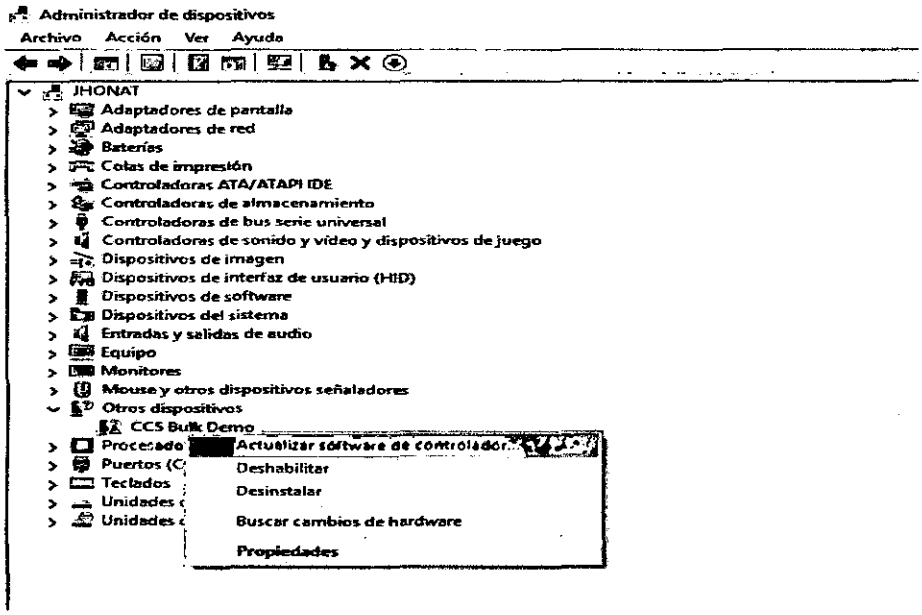


Figura 5. 22 Carga del driver de microchip hacia nuestra tarjeta de ADQ.

4) Luego seleccionamos la opción **"Buscar software del controlador en equipo"**.

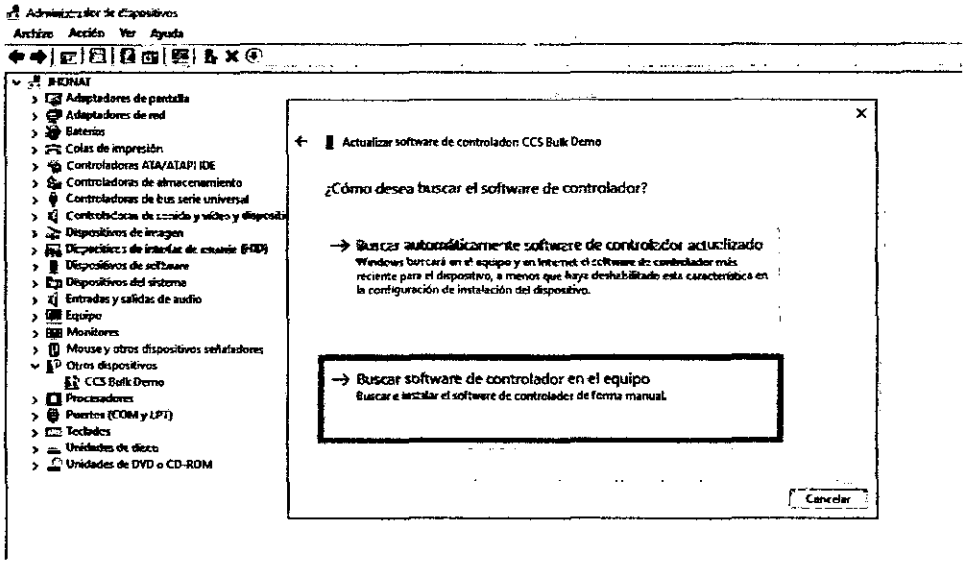


Figura 5. 23 Selección de ruta de ubicación del driver de forma manual.

5) Paso siguiente es buscar la ubicación en donde se encuentra instalado nuestro driver, para lo cual damos clic en el botón "**Examinar**" y luego buscamos la ubicación nombrada en el paso 1 para seleccionar el driver. Una vez ubicada la ruta damos clic al botón "**Aceptar**" y luego clic al botón "**Siguiente**".

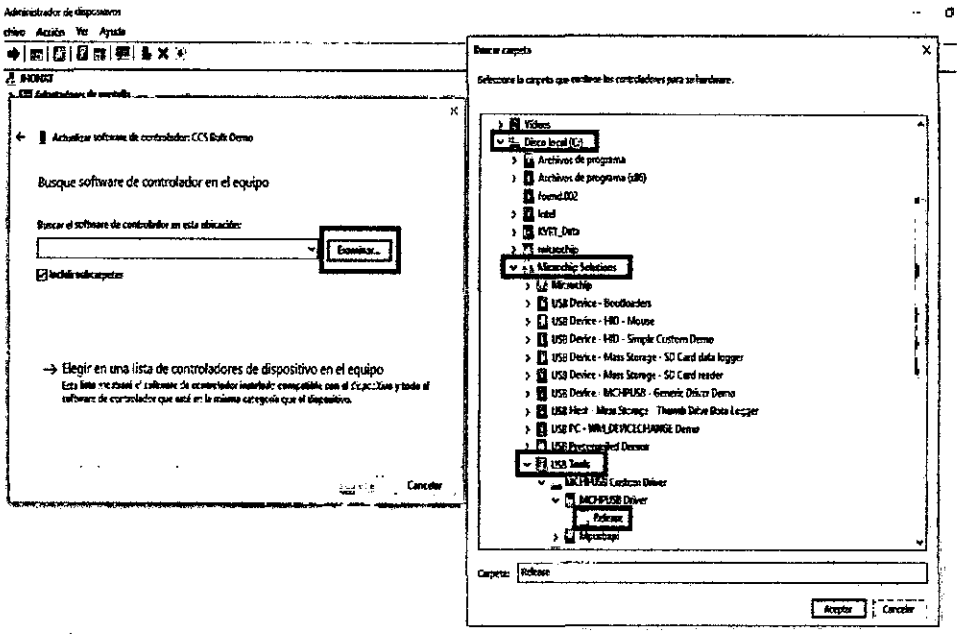


Figura 5. 24 Selección la ubicación del driver de microchip a instalar

6) Luego se cargara el driver y nos aparecerá la siguiente pantalla.

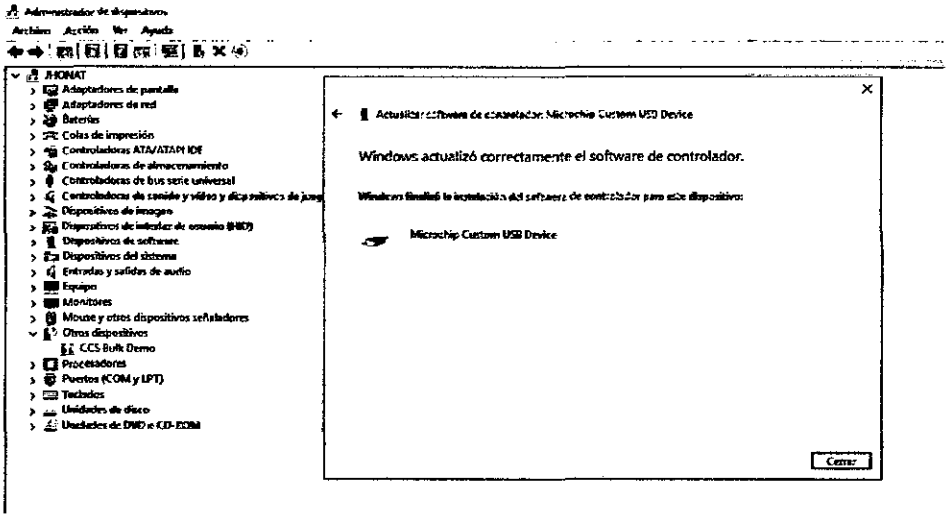


Figura 5. 25 Instalación exitosa del software del controlador.

7) Damos clic al botón "**Cerrar**" y ahora podemos ver que nuestra tarjeta de ADQ está instalada y lista para empezar a realizar la comunicación.

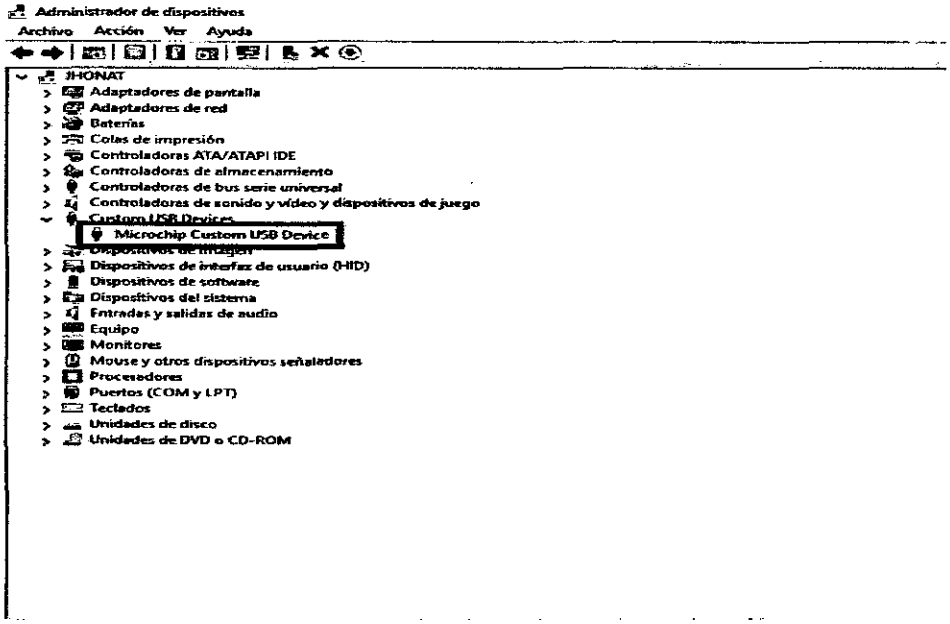


Figura 5. 26 Tarjeta ADQ instalada y lista para establecer comunicación

5.9. Descripción de la Interfaz de comunicaciones

La interfaz de comunicaciones fue desarrollada en el software Matlab 2014 en la versión de 32 bits, ya que la librería para comunicación mphusbapi descrita anteriormente solo trabaja para esta versión; mas no la de 64 bits, por lo tanto debemos tener en cuenta este detalle al momento de desarrollar nuestro programa.

Para poder desarrollar nuestra interfaz de comunicaciones, tenemos que acceder al entorno de programación visual disponible en Matlab llamada GUIDE; también conocida como interfaz gráfica de usuario, la cual nos proporciona todo un conjunto de herramientas necesarias para simplificar el proceso de diseño y creación del interfaz gráfico. En la ventana grafica tenemos a disposición paneles, botones, cuadros de texto, barras de desplazamiento, frames, etc. Genera automáticamente, dos ficheros uno con extensión **.fig** que contiene la información sobre el aspecto visual del interface y otro fichero **.m** en el que se codifica la respuesta a las acciones del usuario sobre los controles.

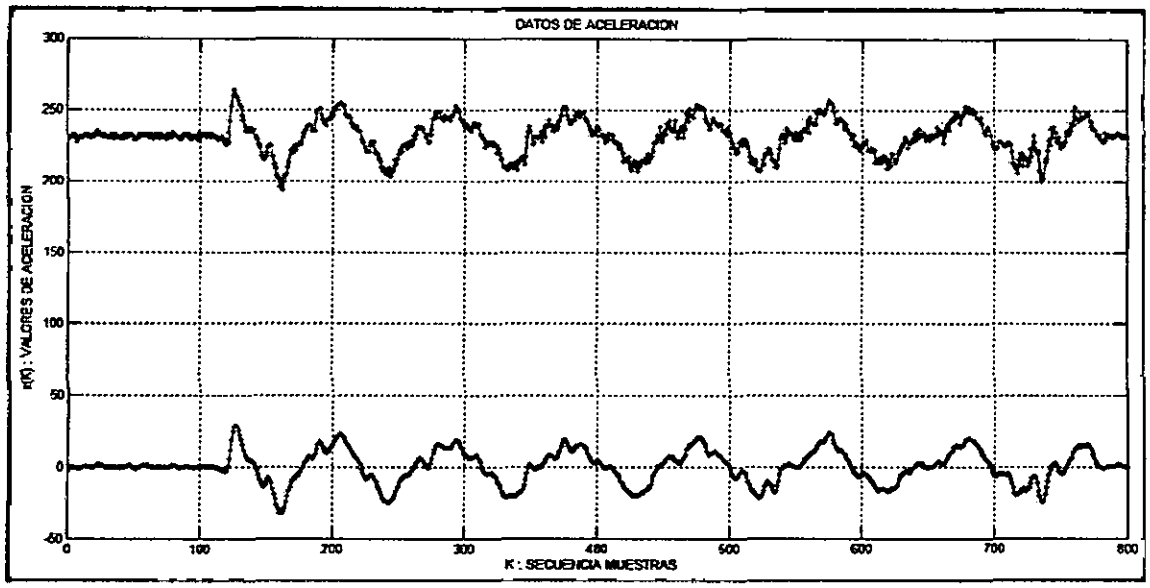


Figura 6. 1 Grafica correspondiente a datos de aceleración del sensor (color azul) y datos de aceleración dinámica (color rojo).

De la figura 6.1 podemos observar que dichos datos contienen "ruido", por lo tanto antes de procesar dichos datos para obtener la velocidad y el desplazamiento del Slave, necesitamos "***suavizar***" (filtrar), estos datos. Este filtrado se realiza mediante la función ***smooth***. Usada de la siguiente manera:

$$\mathbf{azs = smooth\ (azd)} \qquad \mathbf{(6.3)}$$

Para obtener la aceleración en [m/s^2], debemos tomar los datos de aceleracion **azs** y multiplicarlos por el factor de escala del acelerómetro igual a SF = 0.04, mediante:

$$\mathbf{az = azs * SF} \quad [m/s^2] \qquad \mathbf{(6.4)}$$

La figura 6.2 muestra la accleración dinámica "**azd**", la aceleración suavizada "**azs**" y la aceleración en unidades de [m/s^2].

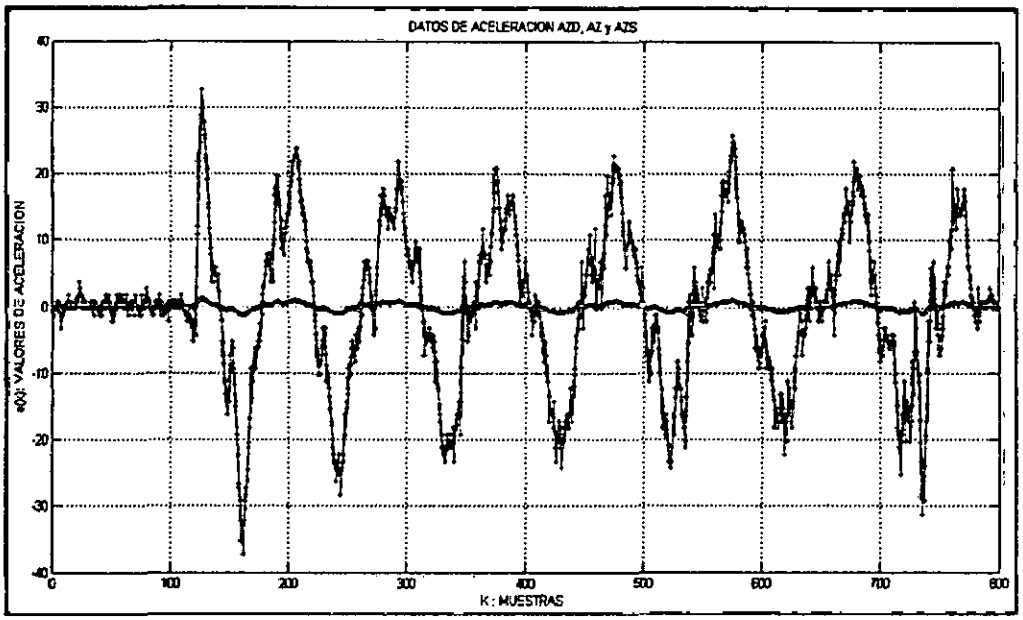


Figura 6. 2 Grafica correspondiente a datos de aceleración dinámica (azul), aceleración suavizada (rojo) y aceleración en [m/s^2]

6.2. Análisis de datos

Para obtener la velocidad y el desplazamiento del Slave, a partir de los datos de aceleración se usaran los datos de aceleración dados en el vector **az,** que contiene los datos de aceleración dinámica del eje z en unidades de [m/s^2], por lo tanto la velocidad estará definida en unidades de [m/s] y el desplazamiento en unidades de [m].

El algoritmo de integración para obtener la velocidad y su respectivo desplazamiento, esta dado mediante la aproximación dada por:

$$y(k) = y(k-1) + 0.5 *(x(k) + x(k-1))*Ts \qquad (6.5)$$

Donde:

Y(k) : secuencia de la integral.

X(k) : secuencia de datos de entrada.

6.2.1. Obtencion de la velocidad y desplazamiento

Usando la formula de integración dada en al ecuación 6.5 y el procedimiento mostrado en el siguiente listado se obtiene la velocidad "v"y el desplazamiento "z"del Slave. Dichos resultados del proceso de integración son mostrados en la figura 6.3.

```
%% calcular velocidad integrando acc
N=length(az);
Ts = 0.04;
v=zeros(N,1);
ki =2;
for k=ki:N
    v(k)=v(k-1)+(az(k)+az(k-1))*0.5*Ts;
end
v1 = detrend(v);

%% calcular distancia y integrando velocidad
z=zeros(N,1);
ki =2;
for k=ki:N
    z(k)=z(k-1)+(v1(k)+v1(k-1))*0.5*0.04;
end

p = polyfit(x,z,2);
f = polyval(p,x);
z1 = y - f;
plot(az,'b.-');grid; hold on;
plot(v1,'r.-');
plot(z1,'g.-');
```

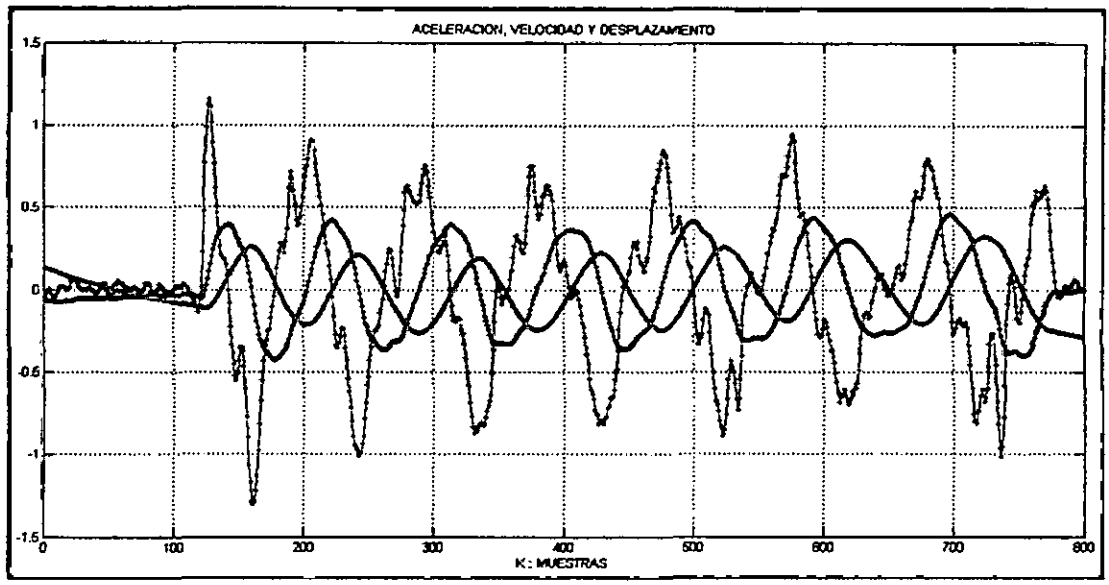


Figura 6. 3 Grafica de la aceleración [m/s²](azul), velocidad [m/s](rojo) y desplazamiento[m] (verde).

El uso de la función **"detrend"** se utiliza para reducir la variación de la velocidad generada por errores constantes en la medición de datos de aceleración que origina una tendencia lineal que crece con el tiempo. Y la utilización de las funciones **"polyfit"** y **"polyval",** se utilizan para reducir los errores de tipo cuadráticos que provienen de los errores lineales de la velocidad.

6.3. Contrastación de hipótesis

A pesar que el proceso de integración de la aceleración inserta un error acumulativo que genera una tendencia lineal a crecer; en el caso de la velocidad, y una tendencia a crecer en forma de parábola; para el desplazamiento, con las técnicas de procesado de señal desarrollado en el apartado anterior es posible finalmente obtener un desplazamiento que se ajusta a la distancia real recorrida por el Slave.

Por lo cual es posible medir el desplazamiento de un tipo de movimiento periodico a partir de datos de aceleración, aplicando el principio fisico que nos dice que la distancia recorrida es igual integrar dos veces la aceleración.

Bibliografía

1. A., A. (s.f.). Obtenido de http://unoscuantospoemas.blogspot.pe/2011/08/comunicacion-pic18fxxxx-con-matlab-por.html

2. Araujo Diaz, D. (2004). *Programacion de Microcontroladores PIC-Microchip.* Mexico D.F.

3. *Buffer de Datos*. (s.f.). Obtenido de https://asm86.wordpress.com/2012/10/18/que-es-un-buffer-de-datos/

4. *El bus USB* . (s.f.). Obtenido de http://www.ciciliani.com.ar/USB/datos_tecnicos/topologia/capafisica.html

5. Garcia Breijo, E. (2008). *Compilador C CCS y simulador Proteus para microcontroladores PIC.* Alfa y Omega.

6. Garcia Bustamante, E. (2013). *Nuevas tendencias en la tecnologia Bluetooth.* Costa Rica.

7. Gomez, P. M. (2008). *USB.*

8. Gonzalo Arribas, M. (2005). *Electronica de potencia y medida.* Oviedo.

9. Humberto Sanz, J., & Salazar Arbelaez, O. (2004). Determinacion de la aceleracion, velocidad y desplazamiento utilizando acelerometros micromaquinados. *Scientia et Technica Año X.*

10. *Implementing Circular/Ring Buffer in Embedded C*. (s.f.). Obtenido de http://embedjournal.com/implementing-circular-buffer-embedded-c/

11. *Ingenieria de Microcontroladores*. (s.f.). Obtenido de www.i-micro.com

12. J., M. (s.f.). *PIC18F2550 y USB, Desarrollo de aplicaciones.*

13. James, D. (2003). Accelerometer design and applications. *Micromachining evangelist, Analog Device.*

14. Larry, H. (2000). *Data Communications.*

15. Lopez Sanchez, J., A. Rojas, F., Trujillo L., C., & A. Guacaneme, J. (s.f.). *Recomendaciones para el diseño de circuitos impresos de potencia.* Francisco Jose de Caldas.

16. Manfred, P., & Arraigada, M. (2006). *Calculation of displacements of measured accelerations,analysis of two accelerometers and application in road.*

17. Manzanares del Moral, A. (2007). *Estudio de modelos matematicos de acelerometros comerciales.* Sevilla.

18. MathWorks. (2015). *Creating Graphical User Interfaces.*

19. Mazas Yarza, M. (Marzo 2007). *Analisis de acelerometros comerciales como sensores para medir desplazamiento.* Zaragoza.

20. Monje Centeno, D. (2010). *Conceptos electronicos en la medida de la aceleracion y de la vibracion.* Sevilla.

21. Moreno Fernandes, A. (2004). *El bus I2C.* Cordova, España.

22. PFC ES Ingenieros. (s.f.). *Comunicaciones con Matlab.*

23. *Protocolos ARQ.* (s.f.). Obtenido de http://labit301.upct.es/~jose/protocolos.html

24. S. Tanenbaum, A. (2003). *Redes de Computadoras.* Pearson.

25. Vega N., F. (s.f.). Obtenido de http://en.calameo.com/read/00261017728ba7c6f683f

```
function varargout = interfaz_tesis(varargin)
% INTERFAZ_TESIS MATLAB code for interfaz_tesis.fig
%      INTERFAZ_TESIS, by itself, creates a new INTERFAZ_TESIS or raises the existing
%      singleton*.
%
%      H = INTERFAZ_TESIS returns the handle to a new INTERFAZ_TESIS or the handle to
%      the existing singleton*.
%
%      INTERFAZ_TESIS('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in INTERFAZ_TESIS.M with the given input arguments.
%
%      INTERFAZ_TESIS('Property','Value',...) creates a new INTERFAZ_TESIS or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before interfaz_tesis_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to interfaz_tesis_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help interfaz_tesis

% Last Modified by GUIDE v2.5 29-Apr-2016 14:51:09

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @interfaz_tesis_OpeningFcn, ...
                   'gui_OutputFcn',  @interfaz_tesis_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before interfaz_tesis is made visible.
function interfaz_tesis_OpeningFcn(hObject, eventdata, handles, varargin)
```

```
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to interfaz_tesis (see VARARGIN)

global t handles1 valor

% Choose default command line output for interfaz_tesis
handles1 = handles;

% Choose default command line output for menu
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

%**************************** Creamos el objeto timer para la ADQ de data de nivel de bateria
**********************************
 t = timer('TimerFcn',@BackGround, 'ExecutionMode', 'fixedSpacing' ,'BusyMode', 'queue'
,'Period', 1);
 stop(t);
%*****************************************************************************************
**************************
 set(handles1.conectar_master_gateway,'enable','on');
 set(handles1.desconectar_master_gateway,'enable','off');
 set(handles1.txt_usb,'BackgroundColor','white');
 set(handles1.txt_usb,'string','DISCONNECTED');
 set(handles1.conectar_gateway_slave,'enable','off');
 set(handles1.desconectar_gateway_slave,'enable','off');
 set(handles1.txt_bt,'BackgroundColor','white');
 set(handles1.txt_bt,'string','DISCONNECTED');
 set(handles1.exit,'enable','on');
 set(handles1.gravedad,'enable','off');
 set(handles1.muestreo,'enable','off');
 set(handles1.stop,'enable','off');
 set(handles1.iniciar_medicion,'enable','off');
 set(handles1.limpiar,'enable','off');
 set(handles1.exportar,'enable','off');
 set(handles1.graficar,'enable','off');

 data_time = fix(clock);
 data_time1 = num2str(data_time(1));
 data_time2 = num2str(data_time(2));
 data_time3 = num2str(data_time(3));
 data_time4 = num2str(data_time(4));
 data_time5 = num2str(data_time(5));
 data_time6 = num2str(data_time(6));
 set(handles1.evento,'string',[data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
 valor = get(handles1.evento,'string');
 set(handles1.evento,'string',[valor,char(13),'INTERFAZ COMUNICACION : START']);
```

```
% UIWAIT makes interfaz_tesis wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = interfaz_tesis_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


%TIMER EMPIEZA A REALIZAR TAREA ASIGNADA
function BackGround(hObject,eventdata)
global conectado t data_in data_out my_in_pipe my_out_pipe vid_pid_norm handles1 valor
fail_bat a gravedad sample max_error time_out_read time_out_write

   [conectado] = calllib('libreria','MPUSBGetDeviceCount',vid_pid_norm);
   if (conectado ==0)
      stop (t);
      set(handles1.conectar_master_gateway,'enable','on');
      set(handles1.desconectar_master_gateway,'enable','off');
      set(handles1.txt_usb,'string','DISCONNECTED');
      set(handles1.txt_usb,'BackgroundColor','red');
      set(handles1.conectar_gateway_slave,'enable','on');
      set(handles1.desconectar_gateway_slave,'enable','off');
      set(handles1.txt_bt,'string','DISCONNECTED');
      set(handles1.txt_bt,'BackgroundColor','red');
      data_time = fix(clock);
      data_time1 = num2str(data_time(1));
      data_time2 = num2str(data_time(2));
      data_time3 = num2str(data_time(3));
      data_time4 = num2str(data_time(4));
      data_time5 = num2str(data_time(5));
      data_time6 = num2str(data_time(6));
      valor = get(handles1.evento,'string');
      set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
      valor = get(handles1.evento,'string');
      set(handles1.evento,'string','CONNECTION MASTER-GATEWAY : LOSS');
      conectado = 0;
      errorbox('CONNECTION MASTER-GATEWAY : LOSS')
   else

      SEC = data_in(3);
      ACK = 6;
      CMD = 'L';
      DATA = 0;
      pkt_bateria = char(SendtoSlave(SEC,ACK,CMD,DATA));
```

```
    data_out = uint8(pkt_bateria);
    calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(7), uint8(7),
uint8(time_out_write)); % Se envia el dato al PIC
    [aa,bb,data_in,dd] = calllib('libreria', 'MPUSBRead',my_in_pipe, data_in, uint8(64), uint8(64),
uint8(time_out_read)); % Se recibe el dato que envia el PIC

    valid_pkt = CheckData(CMD,data_in,dd);

    switch valid_pkt
      case 0 %pkt valido
        data1 = char(data_in(10));
        data2 = char(data_in(11));
        data3 = char(data_in(12));
        data4 = char(data_in(13));
        g_actual = data_in(14);
        f_actual = data_in(15);

        data = strcat(data1,data2,data3,data4);
        data_level = str2double(data)*2;
        data_level_battery = num2str(data_level);
        set(handles1.txt_level,'string',[data_level_battery,'V']);

        if(data_level >= 8.00)
          set(handles1.txt_level,'BackgroundColor','green');

        elseif(data_level > 7.50 && data_level < 8.00)
          set(handles1.txt_level,'BackgroundColor','yellow');

        elseif(data_level <= 7.50 )
          set(handles1.txt_level,'BackgroundColor','red');
          a = a - 1;
          if a == 57 || a == 25
          data_time = fix(clock);
          data_time1 = num2str(data_time(1));
          data_time2 = num2str(data_time(2));
          data_time3 = num2str(data_time(3));
          data_time4 = num2str(data_time(4));
          data_time5 = num2str(data_time(5));
          data_time6 = num2str(data_time(6));
          valor = get(handles1.evento,'string');
          set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
          valor = get(handles1.evento,'string');
          set(handles1.evento,'string',[valor,char(13),'NIVEL BATERIA : BAJA']);
          warndlg('NIVEL BATERIA : BAJA - GUARDE LOS DATOS Y REEMPLAZE LA
BATERIA');
          if a == 0
            a=60;
          end
          end
        end

        switch g_actual
```

```
            case 1
               set(handles1.txt_gravedad,'string','2g');
            case 2
               set(handles1.txt_gravedad,'string','4g');
            case 3
               set(handles1.txt_gravedad,'string','8g');
            case 4
               set(handles1.txt_gravedad,'string','16g');
         end

         if(g_actual == gravedad)
            set(handles1.txt_gravedad,'BackgroundColor','green');
         else
            set(handles1.txt_gravedad,'BackgroundColor','red');
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'GRAVITY ACTUAL DIFFERENT THE
SELECTED']);
            msgbox('GRAVITY ACTUAL DIFFERENT');
            pause(3);
         end

         switch f_actual
            case 1
               set(handles1.txt_sample,'string','12.5Hz');
            case 2
               set(handles1.txt_sample,'string','25Hz');
            case 3
               set(handles1.txt_sample,'string','50Hz');
            case 4
               set(handles1.txt_sample,'string','100Hz');

         end

         if(f_actual == sample)
            set(handles1.txt_sample,'BackgroundColor','green');
         else
            set(handles1.txt_sample,'BackgroundColor','red');
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
```

```
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'FRECUENCY ACTUAL DIFFERENT
THE SELECTED']);
            msgbox('FRECUENCY ACTUAL DIFFERENT');
            pause(3);
          end

        case 1%pkt con error
          fail_bat = fail_bat+1;
          if fail_bat == max_error
            fail_bat = 0;
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'MAXIMUM ERROR RX PKT
BATTERY']);
            msgbox('RESTART,CHECK CONNECTION GATEWAY - SLAVE');
            stop(t);
          else
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'ERROR RX PKT BATTERY']);
            errordlg('ERROR RX PKT BATTERY');
          end

        case 2%pkt no recibido
          fail_bat = fail_bat+1;
          if fail_bat == max_error
            fail_bat = 0;
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
```

```
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'MAXIMUM TIMEOUT WAITING PKT
BATTERY']);
            msgbox('RESTART,CONNECTION GATEWAY - SLAVE LOSS');
            stop(t);
          else
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'PKT BATTERY NOT RECEIVED']);
            errordlg('PKT BATTERY NOT RECEIVED');
          end
      end
    end

%guidata(hObject, handles1);


% --- Executes on button press in conectar_master_gateway.
function conectar_master_gateway_Callback(hObject, eventdata, handles)
% hObject    handle to conectar_master_gateway (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of conectar_master_gateway

global t data_in data_out vid_pid_norm out_pipe in_pipe conectado my_in_pipe my_out_pipe
valor handles1 a sample gravedad SEC ACK DATA
global fail_bat fail_connect_slave fail_disconnect_slave max_error fail_sample fail_stop
fail_gravedad fail_acc time_out_write time_out_read
global time_out_read_acc

time_out_write = 30;
time_out_read = 150;
time_out_read_acc = 150;
fail_bat = 0;
fail_connect_slave = 0;
fail_disconnect_slave = 0;
```

```
fail_sample = 0;
fail_stop = 0;
fail_gravedad = 0;
fail_acc = 0;
max_error = 4;
gravedad = 4;
sample = 4;
a = 60;
SEC = 0;
ACK = 6;
DATA = 0;

  % Estado inicial sin conexion del dispositivo
  conectado = 0;
  %**************************** Definimos buffer de entrada y salida
****************************************

  data_in = eye(1,64,'uint8');      % Se declara el vector de datos de entrada (el que se recibe del
PIC)
  data_out = eye(1,64,'uint8');     % Se declara el vector de datos de salida (el que se envia al PIC)
                        % TODOS LOS DATOS SE DECLARAN COMO
                        % UINT8 de lo contrario no hay
                        % comunicación.

  %************************ Establecemos el EP1 para la comunicacion
****************************************

  vid_pid_norm = libpointer('int8Ptr',[uint8('vid_04d8&pid_000b') 0]);
  out_pipe = libpointer('int8Ptr',[uint8('\MCHP_EP1') 0]);
  in_pipe = libpointer('int8Ptr',[uint8('\MCHP_EP1') 0]);

  %************************** Cargamos las librerias para establecer la comunicacion
**********************

  warning off MATLAB:loadlibrary:TypeNotFound % Se desactivan los mensajes de warning al
cargar la librería.

  [notfound, warnings] = loadlibrary ('mpusbapi', '_mpusbapi.h', 'alias', 'libreria');

  % Los archivos _mpusbapi.c y mpusbapi.dll deben de estar en la misma
  % carpeta de trabajo y se obtienen de la descarga del driver en la
  % pagina de microchip ("Microchip MCHPFSUSB v2.4 Installer.zip"),
  % al instalarse queda ubicado en X:\Microchip Solutions\USB
  % Tools\MCHPUSB Custom Driver\Mpusbapi\Dll\Borland_C, en caso de descargar
  % una version de driver más reciente, reemplace éstos archivos por los más
  % nuevos.

  %libisloaded libreria          % Confirma que la librería ha sido
                        % cargada
  %libfunctions('libreria', '-full')  % Muestra en la línea de comandos las
                        % funciones de la librería
  %libfunctionsview libreria       % Muestra en un cuadro lo mismo que la
                        % instrucción anterior
```

```
%calllib('libreria','MPUSBGetDLLVersion');

while (conectado == 0)
  [conectado] = calllib('libreria','MPUSBGetDeviceCount',vid_pid_norm);
  if (conectado ==0)
    selection = questdlg('The card of AQD not found',...
                'Gateway not found',...
                'Search Again','Cancel','Cancel');
    switch selection,
      case 'Cancel',
        data_time = fix(clock);
        data_time1 = num2str(data_time(1));
        data_time2 = num2str(data_time(2));
        data_time3 = num2str(data_time(3));
        data_time4 = num2str(data_time(4));
        data_time5 = num2str(data_time(5));
        data_time6 = num2str(data_time(6));
        valor = get(handles1.evento,'string');
        set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
        valor = get(handles1.evento,'string');
        set(handles1.evento,'string','CONNECTION MASTER-GATEWAY : FAIL');
        return

      case 'Search Again'
        conectado = 0;
        data_time = fix(clock);
        data_time1 = num2str(data_time(1));
        data_time2 = num2str(data_time(2));
        data_time3 = num2str(data_time(3));
        data_time4 = num2str(data_time(4));
        data_time5 = num2str(data_time(5));
        data_time6 = num2str(data_time(6));
        valor = get(handles1.evento,'string');
        set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
        valor = get(handles1.evento,'string');
        set(handles1.evento,'string','CONNECTION MASTER-GATEWAY : SEARCHING');

    end
  end
end

%********** Si el Gateway esta conectado iniciamos la comunicacion mediante la
siguiente secuencia
% Es importante seguir ésta secuencia para comunicarse con el PIC:
% 1. Abrir tuneles
% 2. Enviar/Recibir dato
% 3. Cerrar tuneles
```

```
  if conectado == 1
    %Habilitamos botones
    set(handles1.conectar_master_gateway,'enable','off');
    set(handles1.desconectar_master_gateway,'enable','on');
    set(handles1.txt_usb,'BackgroundColor','green');
    set(handles1.txt_usb,'string','CONNECTED');
    set(handles1.txt_bt,'BackgroundColor','white');
    set(handles1.txt_bt,'string','DISCONNECTED');
    set(handles1.conectar_gateway_slave,'enable','on');
    set(handles1.desconectar_gateway_slave,'enable','on');


    [my_out_pipe] = calllib('libreria', 'MPUSBOpen',uint8 (0), vid_pid_norm, out_pipe, uint8(0),
uint8 (0)); % Se abre el tunel de envio
    [my_in_pipe] = calllib('libreria', 'MPUSBOpen',uint8 (0), vid_pid_norm, in_pipe, uint8 (1),
uint8 (0)); % Se abre el tunel de recepción
    [cdata,map] = imread('usb1.jpg');
    msgbox('GATEWAY CONNECTED','CONNECTION USB','custom',cdata,map);

    data_time = fix(clock);
    data_time1 = num2str(data_time(1));
    data_time2 = num2str(data_time(2));
    data_time3 = num2str(data_time(3));
    data_time4 = num2str(data_time(4));
    data_time5 = num2str(data_time(5));
    data_time6 = num2str(data_time(6));
    valor = get(handles1.evento,'string');
    set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
    valor = get(handles1.evento,'string');
    set(handles1.evento,'string',[valor,char(13),'CONNECTION MASTER-GATEWAY : OK']);
    pause(3);
    msgbox('COMMUNICATE GATEWAY-SLAVE');
  end


  % --- Executes on button press in desconectar_master_gateway.
function desconectar_master_gateway_Callback(hObject, eventdata, handles)
% hObject    handle to desconectar_master_gateway (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global t my_in_pipe my_out_pipe conectado handles1 valor


%********** Cerramos la comunicacion **********
    selection = questdlg('You want to end communication?',...
                'Confirmation',...
                'Yes','No','Yes');
    switch selection,
      case 'Yes',
       stop (t);
       if conectado == 1
```

```
            calllib('libreria', 'MPUSBClose', my_in_pipe);  % Se cierra el tunel de recepción
            calllib('libreria', 'MPUSBClose', my_out_pipe); % Se cierra el tunel de envio
        end
        unloadlibrary libreria  % Importante descargar la librería de memoria, de lo contrario
genera errores
        set(handles1.conectar_master_gateway,'enable','on');
        set(handles1.desconectar_master_gateway,'enable','off');
        set(handles1.conectar_gateway_slave,'enable','off');
        set(handles1.desconectar_gateway_slave,'enable','off');
        set(handles1.txt_usb,'string','DISCONNECTED');
        set(handles1.txt_usb,'BackgroundColor','white');
        set(handles1.exit,'enable','on');
        set(handles1.gravedad,'enable','off');
        set(handles1.muestreo,'enable','off');
        set(handles1.stop,'enable','off');
        set(handles1.iniciar_medicion,'enable','off');
        set(handles1.guardar,'enable','off');
        set(handles1.limpiar,'enable','off');
        set(handles1.exportar,'enable','off');
        set(handles1.graficar,'enable','off');

        data_time = fix(clock);
        data_time1 = num2str(data_time(1));
        data_time2 = num2str(data_time(2));
        data_time3 = num2str(data_time(3));
        data_time4 = num2str(data_time(4));
        data_time5 = num2str(data_time(5));
        data_time6 = num2str(data_time(6));
        valor = get(handles1.evento,'string');
        set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
        valor = get(handles1.evento,'string');
        set(handles1.evento,'string',[valor,char(13),'DISCONNECTION MASTER-GATEWAY :
OK']);
      case 'No'
        return
    end

    % --- Executes on button press in conectar_gateway_slave.
function conectar_gateway_slave_Callback(hObject, eventdata, handles)
% hObject    handle to conectar_gateway_slave (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global conectado t data_in data_out my_in_pipe my_out_pipe SEC ACK DATA a vid_pid_norm
valor fail_connect_slave handles1 max_error time_out_write
time_out_read_bt=30;
a=60;
[conectado] = calllib('libreria','MPUSBGetDeviceCount',vid_pid_norm);
  if (conectado ==0)
    errordlg('THE GATEWAY NOT FOUND','ERROR CONNECCTION ')
    stop (t);
    set(handles1.conectar_master_gateway,'enable','on');
```

```
    set(handles1.desconectar_master_gateway,'enable','off');
    set(handles1.txt_usb,'string','DISCONNECTED');
    set(handles1.txt_usb,'BackgroundColor','white');
    set(handles1.conectar_gateway_slave,'enable','off');
    set(handles1.desconectar_gateway_slave,'enable','off');
    set(handles1.txt_bt,'string','DISCONNECTED');
    set(handles1.txt_bt,'BackgroundColor','white');
    data_time = fix(clock);
    data_time1 = num2str(data_time(1));
    data_time2 = num2str(data_time(2));
    data_time3 = num2str(data_time(3));
    data_time4 = num2str(data_time(4));
    data_time5 = num2str(data_time(5));
    data_time6 = num2str(data_time(6));
    valor = get(handles1.evento,'string');
    set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
    valor = get(handles1.evento,'string');
    set(handles1.evento,'string','CONNECTION MASTER-GATEWAY : LOSS');
    conectado = 0;
    errorbox('CONNECTION MASTER-GATEWAY : LOSS');
  else


      CMD = 'T';
      %DATA = 0;
      pkt_test = char(SendtoSlave(SEC,ACK,CMD,DATA));
      data_out = uint8(pkt_test);
      calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(7), uint8(7),
uint8(time_out_write)); % Se envia el dato al PIC
      pause(11); % pausa necesaria para recibir una respuesta y verificar si el BT del slave se
conecto correctamente
      [aa,~,data_in,dd] = calllib('libreria', 'MPUSBRead',my_in_pipe, data_in, uint8(64),
uint8(64), uint8(time_out_read_bt));% Se recibe el dato que envia el PIC
      valid_pkt = CheckData(CMD,data_in,dd);

      switch valid_pkt
        case 0 %pkt valido
          set(handles1.desconectar_gateway_slave,'enable','on');
          set(handles1.conectar_gateway_slave,'enable','off');
          set(handles1.txt_bt,'BackgroundColor',[0.0 0.6 1.0]);
          set(handles1.txt_bt,'string','CONNECTED');
          set(handles1.gravedad,'enable','on');
          set(handles1.muestreo,'enable','on');
          set(handles1.stop,'enable','on');
          set(handles1.iniciar_medicion,'enable','on');
          set(handles1.limpiar,'enable','on');
          set(handles1.exportar,'enable','on');
          set(handles1.graficar,'enable','on');

          [cdata,map] = imread('bt.jpg');
          msgbox('CONNECTION MASTER - SLAVE ESTABLISHED','CONNECTION
BT','custom',cdata,map);
```

```
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'CONNECTION GATEWAY-SLAVE :
OK']);
            pause(1);
            start(t);

          case 1%pkt con error
            stop(t);
            fail_connect_slave = fail_connect_slave+1;
            if fail_connect_slave == max_error
              fail_connect_slave = 0;
              data_time = fix(clock);
              data_time1 = num2str(data_time(1));
              data_time2 = num2str(data_time(2));
              data_time3 = num2str(data_time(3));
              data_time4 = num2str(data_time(4));
              data_time5 = num2str(data_time(5));
              data_time6 = num2str(data_time(6));
              valor = get(handles1.evento,'string');
              set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
              set(handles1.txt_bt,'BackgroundColor','red');
              set(handles1.txt_bt,'string','DISCONNECTED');
              valor = get(handles1.evento,'string');
              set(handles1.evento,'string',[valor,char(13),'CONNECTION GATEWAY-SLAVE :
IMPOSSIBLE CONNECT']);
              msgbox('IMPOSSIBLE CONNECT,DISCONNECT GATEWAY AND
RESTART');
            else
              set(handles1.txt_bt,'BackgroundColor','red');
              set(handles1.txt_bt,'string','DISCONNECTED');
              data_time = fix(clock);
              data_time1 = num2str(data_time(1));
              data_time2 = num2str(data_time(2));
              data_time3 = num2str(data_time(3));
              data_time4 = num2str(data_time(4));
              data_time5 = num2str(data_time(5));
              data_time6 = num2str(data_time(6));
              valor = get(handles1.evento,'string');
              set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
              valor = get(handles1.evento,'string');
```

```
            set(handles1.evento,'string',[valor,char(13),'CONNECTION GATEWAY-SLAVE :
FAIL']);
            errordlg('FAILURE TO CONNECT, TRY AGAIN');
          end

        case 2%pkt no recibido
          stop(t);
          fail_connect_slave = fail_connect_slave+1;
          if fail_connect_slave == max_error
            fail_connect_slave = 0;
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            set(handles1.txt_bt,'BackgroundColor','red');
            set(handles1.txt_bt,'string','DISCONNECTED');
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'MAXIMUM ERROR RX PKT
CONNECTION']);
            msgbox('CHECK CONNECTION GATEWAY - SLAVE');
          else
            set(handles1.txt_bt,'BackgroundColor','red');
            set(handles1.txt_bt,'string','DISCONNECTED');
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'PKT CONNECTION NOT
RECEIVED']);
            errordlg('PKT CONNECTION NOT RECEIVED');
          end
      end

  end

%guidata(hObject, handles1);
```

```
% --- Executes on button press in desconectar_gateway_slave.
function desconectar_gateway_slave_Callback(hObject, eventdata, handles)
% hObject    handle to desconectar_gateway_slave (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global conectado t data_in data_out my_in_pipe my_out_pipe SEC DATA ACK vid_pid_norm
valor handles1 max_error fail_disconnect_slave time_out_write time_out_read

stop(t);

[conectado] = calllib('libreria','MPUSBGetDeviceCount',vid_pid_norm);
   if (conectado ==0)
      errordlg('THE GATEWAY NOT FOUND','ERROR CONNECCTION ')
      stop (t);
      set(handles1.conectar_master_gateway,'enable','on');
      set(handles1.desconectar_master_gateway,'enable','off');
      set(handles1.txt_usb,'string','DISCONNECTED');
      set(handles1.txt_usb,'BackgroundColor','white');
      set(handles1.conectar_gateway_slave,'enable','on');
      set(handles1.desconectar_gateway_slave,'enable','off');
      set(handles1.txt_bt,'string','DISCONNECTED');
      set(handles1.txt_bt,'BackgroundColor','white');
      msgbox('CONNECTION MASTER-GATEWAY : LOSS')
      data_time = fix(clock);
      data_time1 = num2str(data_time(1));
      data_time2 = num2str(data_time(2));
      data_time3 = num2str(data_time(3));
      data_time4 = num2str(data_time(4));
      data_time5 = num2str(data_time(5));
      data_time6 = num2str(data_time(6));
      valor = get(handles1.evento,'string');
      set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
      valor = get(handles1.evento,'string');
      set(handles1.evento,'string','CONNECTION MASTER-GATEWAY : LOSS');
      conectado = 0;
   else

         CMD = 'V';
         DATA = 0;
         pkt_test = char(SendtoSlave(SEC,ACK,CMD,DATA));
         data_out = uint8(pkt_test);
         calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(7), uint8(7),
uint8(time_out_write)); % Se envia el dato al PIC
         [aa,bb,data_in,dd] = calllib('libreria', 'MPUSBRead',my_in_pipe, data_in, uint8(64),
uint8(64), uint8(time_out_read)); % Se recibe el dato que envia el PIC
         valid_pkt = CheckData(CMD,data_in,dd);

         switch valid_pkt
            case 0 %pkt valido
               ACK = 6;
               SEC = 0;
               DATA = 0;
```

```
            set(handles1.desconectar_gateway_slave,'enable','off');
            set(handles1.conectar_gateway_slave,'enable','on');
            set(handles1.txt_bt,'BackgroundColor','white');
            set(handles1.txt_bt,'string','DISCONNECTED');
            set(handles1.gravedad,'enable','off');
            set(handles1.muestreo,'enable','off');
            set(handles1.stop,'enable','off');
            set(handles1.iniciar_medicion,'enable','off');
            set(handles1.limpiar,'enable','off');
            set(handles1.exportar,'enable','off');
            set(handles1.graficar,'enable','off');

            [cdata,map] = imread('bt.jpg');
            msgbox('DISCONNECTION MASTER - SLAVE SUCCESSFULL','CONNECTION
BT','custom',cdata,map);
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'DISCONNECTION GATEWAY-SLAVE
: OK']);

          case 2%pkt no recibido
            stop(t);
            fail_disconnect_slave = fail_disconnect_slave+1;
            if fail_disconnect_slave == max_error
              fail_disconnect_slave = 0;
              data_time = fix(clock);
              data_time1 = num2str(data_time(1));
              data_time2 = num2str(data_time(2));
              data_time3 = num2str(data_time(3));
              data_time4 = num2str(data_time(4));
              data_time5 = num2str(data_time(5));
              data_time6 = num2str(data_time(6));
              valor = get(handles1.evento,'string');
              set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
              set(handles1.txt_bt,'BackgroundColor','red');
              set(handles1.txt_bt,'string','UNANSWERED');
              valor = get(handles1.evento,'string');
              set(handles1.evento,'string',[valor,char(13),'DISCONNECTION GATEWAY-
SLAVE : IMPOSSIBLE DISCONNECT']);
              msgbox('IMPOSSIBLE DISCONNECT,DISCONNECT GATEWAY AND
RESTART');
            else
              set(handles1.txt_bt,'BackgroundColor','white');
```

```
                set(handles1.txt_bt,'string','UNANSWERED');
                data_time = fix(clock);
                data_time1 = num2str(data_time(1));
                data_time2 = num2str(data_time(2));
                data_time3 = num2str(data_time(3));
                data_time4 = num2str(data_time(4));
                data_time5 = num2str(data_time(5));
                data_time6 = num2str(data_time(6));
                valor = get(handles1.evento,'string');
                set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
                valor = get(handles1.evento,'string');
                set(handles1.evento,'string',[valor,char(13),'DISCONNECTION GATEWAY-
SLAVE : FAIL']);
                errordlg('FAILURE TO DISCONNECT, TRY AGAIN');
              end
          end
    end

 %   guidata(hObject, handles1);

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1


% --- Executes on button press in iniciar_medicion.
function iniciar_medicion_Callback(hObject, eventdata, handles)
% hObject    handle to iniciar_medicion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global conectado t SEC ACK DATA data_in data_out my_in_pipe my_out_pipe valor handles1
m_sa max_error muestras fail_init fail_stop time_out_write time_out_read
global time_out_read_acc axis sample

stop (t);
delay_f1 = 0.3;      %tiempo de espera para garantizar 5 datos en el buffer para Fm = 12.5Hz
delay_f2 = 0.115;    %tiempo de espera para garantizar 5 datos en el buffer para Fm = 25Hz
secuencia = 0;
error = 0;
error_pkt = 0;
num_muestras = get(handles1.num_data,'string');
muestras = str2double(num_muestras); % numero de datos a tomar definidos por el usuario
m_sa = zeros(muestras,3); % matriz de almacenamiento de data

if conectado == 1

   SEC = 0;
```

```
   ACK = 6;
   CMD = 'M';
   DATA = 0;
   pkt_inicio = char(SendtoSlave(SEC,ACK,CMD,DATA))
   SEC = pkt_inicio(3);
   data_out = uint8(pkt_inicio);
   calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(7), uint8(7),
uint8(time_out_write)) % Se envia el dato al PIC

   %delay para esperar paquete de respuesta del slave asegurando la escritura de 10 datos en el
buffer circular
   switch sample
      case 1  %Fm=12.5hz
         pause(1);
      case 2  %Fm=25hz
         pause(0.6);
   end

   [aa,bb,data_in,dd] = calllib('libreria', 'MPUSBRead',my_in_pipe, data_in, uint8(64), uint8(64),
uint8(time_out_read)) % Se recibe el dato que envia el PIC

   valid_pkt = CheckData(CMD,data_in,dd);

      switch valid_pkt
         case 0 %pkt valido
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'ACCELEROMETER MODE :
MEASUREMENT']);
            %pause(0.64);    % delay para estabilizar acelerometro
            %pause()
               SEC = 0;
               ACK = 6;
               CMD = 'A';
               DATA = axis;
               pkt_acc = char(SendtoSlave(SEC,ACK,CMD,DATA));
               seq_ant = SEC;
               data_out = uint8(pkt_acc);

            while secuencia < muestras
               calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(7), uint8(7),
uint8(time_out_write)) % Se envia el dato al PIC
               [aa,bb,data_in,dd] = calllib('libreria', 'MPUSBRead',my_in_pipe, data_in, uint8(64),
uint8(64), uint8(time_out_read_acc)) % Se recibe el dato que envia el PIC
```

```
if dd == 0 || error_pkt == 5
  error = error + 1;
end

[rpta,pkt] = ReadFromSlave(data_in,seq_ant,dd,error);

if error == max_error
  error = 0;
end

pkt_acc = char(SendtoSlave(pkt(1),pkt(2),pkt(3),axis));
data_out = uint8(pkt_acc);
seq_ant = pkt(1);
SEC = seq_ant;

switch rpta
  case 0 %data correcta
    seq_dat = char(data_in(5),data_in(6),data_in(7),data_in(8),data_in(9))';
    secuencia = str2double(seq_dat)-4;

    eje =
char(data_in(10),data_in(11),data_in(12),data_in(13),data_in(14),data_in(15))';
    axis_acc = str2double(eje);

    pointer_iw = char(data_in(40),data_in(41))';
    pointer_num_iw = str2double(pointer_iw);

    pointer_ir = char(data_in(42),data_in(43))';
    pointer_num_ir = str2double(pointer_ir);

    pointer = pointer_num_iw - pointer_num_ir;

    v_sa = [secuencia axis_acc pointer];
    m_sa(secuencia,:) = v_sa;
    assignin('base','data_acc',m_sa);

    secuencia = secuencia+1;
    eje =
char(data_in(16),data_in(17),data_in(18),data_in(19),data_in(20),data_in(21))';
    axis_acc = str2double(eje);

    v_sa = [secuencia axis_acc pointer];
    m_sa(secuencia,:) = v_sa;
    assignin('base','data_acc',m_sa);

    secuencia = secuencia+1;
    eje =
char(data_in(22),data_in(23),data_in(24),data_in(25),data_in(26),data_in(27))';
    axis_acc = str2double(eje);

    v_sa = [secuencia axis_acc pointer];
    m_sa(secuencia,:) = v_sa;
```

```
                assignin('base','data_acc',m_sa);

                secuencia = secuencia+1;
                eje =
char(data_in(28),data_in(29),data_in(30),data_in(31),data_in(32),data_in(33))';
                axis_acc = str2double(eje);

                v_sa = [secuencia axis_acc pointer];
                m_sa(secuencia,:) = v_sa;
                assignin('base','data_acc',m_sa);

                secuencia = secuencia+1;
                eje =
char(data_in(34),data_in(35),data_in(36),data_in(37),data_in(38),data_in(39))';
                axis_acc = str2double(eje);

                v_sa = [secuencia axis_acc pointer];
                m_sa(secuencia,:) = v_sa;
                assignin('base','data_acc',m_sa);

                switch axis
                  case 1
                    set(handles1.txt_secuencia,'string',seq_dat);
                    set(handles1.txt_ejex,'string',eje);

                  case 2
                    set(handles1.txt_secuencia,'string',seq_dat);
                    set(handles1.txt_ejey,'string',eje);

                  case 3
                    set(handles1.txt_secuencia,'string',seq_dat);
                    set(handles1.txt_ejez,'string',eje);
                end

                %delay para asegurar que siempre el buffer tenga datos disponibles
                 switch sample
                  case 1  %Fm=25hz
                    pause(delay_f1);
                  case 2  %Fm=50hz
                    pause(delay_f2);
                  case 3  %Fm=100hz
                    pause(delay_f3);
                end

              case 1 %transmicion correcta y fin del bucle
                secuencia = muestras; %condicion para salir del bucle
                set(handles1.gravedad,'enable','on');
                set(handles1.muestreo,'enable','on');

              case 2 % Buffer empty
                data_time = fix(clock);
                data_time1 = num2str(data_time(1));
                data_time2 = num2str(data_time(2));
```

```
                    data_time3 = num2str(data_time(3));
                    data_time4 = num2str(data_time(4));
                    data_time5 = num2str(data_time(5));
                    data_time6 = num2str(data_time(6));
                    valor = get(handles1.evento,'string');

set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
                    valor = get(handles1.evento,'string');
                    set(handles1.evento,'string',[valor,char(13),'BUFFER EMPTY']);
                    errordlg('!!Buffer Empty!!','Error Write');
                    secuencia = muestras;

                  case 3 % Buffer Full
                    data_time = fix(clock);
                    data_time1 = num2str(data_time(1));
                    data_time2 = num2str(data_time(2));
                    data_time3 = num2str(data_time(3));
                    data_time4 = num2str(data_time(4));
                    data_time5 = num2str(data_time(5));
                    data_time6 = num2str(data_time(6));
                    valor = get(handles1.evento,'string');

set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
                    valor = get(handles1.evento,'string');
                    set(handles1.evento,'string',[valor,char(13),'BUFFER FULL']);
                    errordlg('!!Buffer Full!!','Error Read');
                    secuencia = muestras;

                  case 4 % Pkt invalido
                    error_pkt = error_pkt + 1;
                    data_time = fix(clock);
                    data_time1 = num2str(data_time(1));
                    data_time2 = num2str(data_time(2));
                    data_time3 = num2str(data_time(3));
                    data_time4 = num2str(data_time(4));
                    data_time5 = num2str(data_time(5));
                    data_time6 = num2str(data_time(6));
                    valor = get(handles1.evento,'string');

set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
                    valor = get(handles1.evento,'string');
                    set(handles1.evento,'string',[valor,char(13),'PKT INVALIDO : NACK']);
                    errordlg('Pkt Invalido : NACK');

                  case 5 % Pkt Repetido
                    error_pkt = error_pkt + 1;
                    data_time = fix(clock);
                    data_time1 = num2str(data_time(1));
                    data_time2 = num2str(data_time(2));
                    data_time3 = num2str(data_time(3));
```

```
                    data_time4 = num2str(data_time(4));
                    data_time5 = num2str(data_time(5));
                    data_time6 = num2str(data_time(6));
                    valor = get(handles1.evento,'string');

set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
                    valor = get(handles1.evento,'string');
                    set(handles1.evento,'string',[valor,char(13),'PKT DESCARTADO :
REPETIDO']);
                    errordlg('Pkt Descartado : Repetido');

                  case 6 % Pkt invalido por error de iniciador/terminador pkt
                    error_pkt = error_pkt + 1;
                    data_time = fix(clock);
                    data_time1 = num2str(data_time(1));
                    data_time2 = num2str(data_time(2));
                    data_time3 = num2str(data_time(3));
                    data_time4 = num2str(data_time(4));
                    data_time5 = num2str(data_time(5));
                    data_time6 = num2str(data_time(6));
                    valor = get(handles1.evento,'string');

set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
                    valor = get(handles1.evento,'string');
                    set(handles1.evento,'string',[valor,char(13),'PKT INVALIDO : ERROR
STX/ETX']);
                    errordlg('Pkt Invalido : Error STX/ETX');

                  case 7 % Pkt invalido por error de chsm
                    error_pkt = error_pkt + 1;
                    data_time = fix(clock);
                    data_time1 = num2str(data_time(1));
                    data_time2 = num2str(data_time(2));
                    data_time3 = num2str(data_time(3));
                    data_time4 = num2str(data_time(4));
                    data_time5 = num2str(data_time(5));
                    data_time6 = num2str(data_time(6));
                    valor = get(handles1.evento,'string');

set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
                    valor = get(handles1.evento,'string');
                    set(handles1.evento,'string',[valor,char(13),'PKT INVALIDO : ERROR CHSM']);
                    errordlg('Pkt Invalido : Error CHSM');

                  case 8 %Timeout!! data no recibida
                    data_time = fix(clock);
                    data_time1 = num2str(data_time(1));
                    data_time2 = num2str(data_time(2));
                    data_time3 = num2str(data_time(3));
                    data_time4 = num2str(data_time(4));
```

```
                    data_time5 = num2str(data_time(5));
                    data_time6 = num2str(data_time(6));
                    valor = get(handles1.evento,'string');

set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
                    valor = get(handles1.evento,'string');
                    set(handles1.evento,'string',[valor,char(13),'TIMEOUT']);
                    errordlg('Timeout');

                case 9 %Timeout limite!! Error en la comunicacion , reiniciar
                    data_time = fix(clock);
                    data_time1 = num2str(data_time(1));
                    data_time2 = num2str(data_time(2));
                    data_time3 = num2str(data_time(3));
                    data_time4 = num2str(data_time(4));
                    data_time5 = num2str(data_time(5));
                    data_time6 = num2str(data_time(6));
                    valor = get(handles1.evento,'string');

set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
                    valor = get(handles1.evento,'string');
                    set(handles1.evento,'string',[valor,char(13),'TIMEOUT LIMIT']);
                    errordlg('!!Timeout limit!!','Error Communication');
                    pause(2);
                    warndlg('Desconectar ADQ y Reiniciar');
                    secuencia = muestras;
                end
            end

        SEC = seq_ant;
        ACK = 6;
        CMD = 'S';
        DATA = 0;
        pkt_acc = char(SendtoSlave(SEC,ACK,CMD,DATA));
        SEC = pkt_acc(3);
        data_out = uint8(pkt_acc);
        calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(7), uint8(7),
uint8(time_out_write)); % Se envia el dato al PIC
        [aa,bb,data_in,dd] = calllib('libreria', 'MPUSBRead',my_in_pipe, data_in, uint8(64),
uint8(64), uint8(time_out_read)); % Se recibe el dato que envia el PIC

        valid_pkt_stop = CheckData(CMD,data_in,dd);

        switch valid_pkt_stop
            case 0 %pkt valido
                set(handles1.graficar,'enable','on');
                set(handles1.limpiar,'enable','on');
                set(handles1.exportar,'enable','on');
                set(handles1.guardar,'enable','on');
                data_time = fix(clock);
                data_time1 = num2str(data_time(1));
```

```
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'ACCELEROMETER MODE :
STANDBY']);
            msgbox('ACCELEROMETER MODE : STANDBY');
            %pause(2);
            %start(t);

          case 1%pkt con error
            fail_stop = fail_stop+1;
            if fail_stop == max_error
              fail_stop = 0;
              data_time = fix(clock);
              data_time1 = num2str(data_time(1));
              data_time2 = num2str(data_time(2));
              data_time3 = num2str(data_time(3));
              data_time4 = num2str(data_time(4));
              data_time5 = num2str(data_time(5));
              data_time6 = num2str(data_time(6));
              valor = get(handles1.evento,'string');
              set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
              valor = get(handles1.evento,'string');
              set(handles1.evento,'string',[valor,char(13),'MAXIMUM ERROR RX PKT STOP']);
              msgbox('RESTART,CHECK CONNECTION GATEWAY - SLAVE');
              %stop(t);
            else
              data_time1 = num2str(data_time(1));
              data_time2 = num2str(data_time(2));
              data_time3 = num2str(data_time(3));
              data_time4 = num2str(data_time(4));
              data_time5 = num2str(data_time(5));
              data_time6 = num2str(data_time(6));
              valor = get(handles1.evento,'string');
              set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
              valor = get(handles1.evento,'string');
              set(handles1.evento,'string',[valor,char(13),'STOP THE MEASUREMENT
ERROR']);
              errordlg('STOP MEASURING ERROR');
              %pause(1);
              %start(t);
            end

          case 2%pkt no recibido
            fail_stop = fail_stop+1;
```

```
            if fail_stop == max_error
              fail_stop = 0;
              data_time = fix(clock);
              data_time1 = num2str(data_time(1));
              data_time2 = num2str(data_time(2));
              data_time3 = num2str(data_time(3));
              data_time4 = num2str(data_time(4));
              data_time5 = num2str(data_time(5));
              data_time6 = num2str(data_time(6));
              valor = get(handles1.evento,'string');
              set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
              valor = get(handles1.evento,'string');
              set(handles1.evento,'string',[valor,char(13),'MAXIMUM TIMEOUT WAITING
PKT STOP']);
              msgbox('RESTART,CONNECTION GATEWAY - SLAVE LOSS');
              %stop(t);
            else
              data_time = fix(clock);
              data_time1 = num2str(data_time(1));
              data_time2 = num2str(data_time(2));
              data_time3 = num2str(data_time(3));
              data_time4 = num2str(data_time(4));
              data_time5 = num2str(data_time(5));
              data_time6 = num2str(data_time(6));
              valor = get(handles1.evento,'string');
              set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
              valor = get(handles1.evento,'string');
              set(handles1.evento,'string',[valor,char(13),'PKT STOP THE MEASUREMENT
NOT RECEIVED']);
              errordlg('PKT STOP THE MEASUREMENT NOT RECEIVED');
              %pause(1);
              %start(t);
            end
        end
          pause(1);
          start(t);

        case 1%pkt con error0
          fail_init = fail_init+1;
          if fail_init == max_error
            fail_init = 0;
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
```

```
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'MAXIMUM ERROR RX PKT START']);
            msgbox('RESTART,CHECK CONNECTION GATEWAY - SLAVE');
            %stop(t);
          else
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'START ERROR PKT']);
            errordlg('START ERROR PKT');
            %pause(2);
            start(t);
          end

        case 2%pkt no recibido
          fail_init = fail_init+1;
          if fail_init == max_error
            fail_init = 0;
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'MAXIMUM TIMEOUT WAITING PKT
START']);
            msgbox('RESTART,CONNECTION GATEWAY - SLAVE LOSS');
            %stop(t);
          else
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
```

```
            set(handles1.evento,'string',[valor,char(13),'START PKT NOT RECEIVED']);
            errordlg('START PKT NOT RECEIVED');
            %pause(2);
            start(t);
          end
    end
end


% --- Executes on button press in stop.
function stop_Callback(hObject, eventdata, handles)
% hObject    handle to stop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global conectado ACK DATA SEC t data_in data_out my_in_pipe my_out_pipe vid_pid_norm
valor fail_stop max_error handles1 time_out_write time_out_read

stop (t);

  [conectado] = calllib('libreria','MPUSBGetDeviceCount',vid_pid_norm);
  if (conectado ==0)
    stop (t);
    set(handles1.conectar_master_gateway,'enable','on');
    set(handles1.desconectar_master_gateway,'enable','off');
    set(handles1.txt_usb,'string','DISCONNECTED');
    set(handles1.txt_usb,'BackgroundColor','red');
    set(handles1.conectar_gateway_slave,'enable','on');
    set(handles1.desconectar_gateway_slave,'enable','off');
    set(handles1.txt_bt,'string','DISCONNECTED');
    set(handles1.txt_bt,'BackgroundColor','red');
    errorbox('CONNECTION MASTER-GATEWAY : LOSS');
    data_time = fix(clock);
    data_time1 = num2str(data_time(1));
    data_time2 = num2str(data_time(2));
    data_time3 = num2str(data_time(3));
    data_time4 = num2str(data_time(4));
    data_time5 = num2str(data_time(5));
    data_time6 = num2str(data_time(6));
    valor = get(handles1.evento,'string');
    set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
    valor = get(handles1.evento,'string');
    set(handles1.evento,'string','CONNECTION MASTER-GATEWAY : LOSS');
    conectado = 0;
  else

    SEC = 0;
    ACK = 6;
    CMD = 'S';
    DATA = 0;
    pkt_stop = char(SendtoSlave(SEC,ACK,CMD,DATA));
    SEC = pkt_stop(3);
    data_out = uint8(pkt_stop);
```

```
    calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(7), uint8(7),
uint8(time_out_write)); % Se envia el dato al PIC
    [aa,bb,data_in,dd] = calllib('libreria', 'MPUSBRead',my_in_pipe, data_in, uint8(64), uint8(64),
uint8(time_out_read)); % Se recibe el dato que envia el PIC
    valid_pkt = CheckData(CMD,data_in,dd);

    switch valid_pkt
      case 0 %pkt valido
        msgbox('ACCELEROMETER MODE : STANDBY');
        data_time = fix(clock);
        data_time1 = num2str(data_time(1));
        data_time2 = num2str(data_time(2));
        data_time3 = num2str(data_time(3));
        data_time4 = num2str(data_time(4));
        data_time5 = num2str(data_time(5));
        data_time6 = num2str(data_time(6));
        valor = get(handles1.evento,'string');
        set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
        valor = get(handles1.evento,'string');
        set(handles1.evento,'string',[valor,char(13),'ACCELEROMETER MODE :
STANDBY']);
        %pause(1);
        start(t);

      case 1%pkt con error
        %stop(t);
        fail_stop = fail_stop+1;
       if fail_stop == max_error
          fail_stop = 0;
          data_time = fix(clock);
          data_time1 = num2str(data_time(1));
          data_time2 = num2str(data_time(2));
          data_time3 = num2str(data_time(3));
          data_time4 = num2str(data_time(4));
          data_time5 = num2str(data_time(5));
          data_time6 = num2str(data_time(6));
          valor = get(handles1.evento,'string');
          set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
          valor = get(handles1.evento,'string');
          set(handles1.evento,'string',[valor,char(13),'MAXIMUM ERROR RX PKT STOP']);
          msgbox('RESTART,CHECK CONNECTION GATEWAY - SLAVE');
        else
          data_time = fix(clock);
          data_time1 = num2str(data_time(1));
          data_time2 = num2str(data_time(2));
          data_time3 = num2str(data_time(3));
          data_time4 = num2str(data_time(4));
          data_time5 = num2str(data_time(5));
          data_time6 = num2str(data_time(6));
          valor = get(handles1.evento,'string');
```

```
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'STOP THE MEASUREMENT ERROR']);
            errordlg('STOP MEASURING ERROR');
            %pause(2);
            start(t);
          end

        case 2%pkt no recibido
          %stop(t);
          fail_stop = fail_stop+1;
          if fail_stop == max_error
            fail_stop = 0;
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'MAXIMUM TIMEOUT WAITING PKT
STOP']);
            msgbox('RESTART,CONNECTION GATEWAY - SLAVE LOSS');

          else
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'PKT STOP THE MEASUREMENT NOT
RECEIVED']);
            errordlg('PKT STOP THE MEASUREMENT NOT RECEIVED');
            %pause(2);
            start(t);
          end
      end
    end
%guidata(hObject, handles);
```

```
% --- Executes on button press in gravedad.
function gravedad_Callback(hObject, eventdata, handles)
% hObject    handle to gravedad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global conectado ACK SEC t data_in data_out my_in_pipe my_out_pipe vid_pid_norm valor
max_error fail_gravedad handles1 gravedad time_out_write time_out_read

stop(t);


   [conectado] = calllib('libreria','MPUSBGetDeviceCount',vid_pid_norm);
   if (conectado ==0)
      stop (t);
      set(handles1.conectar_master_gateway,'enable','on');
      set(handles1.desconectar_master_gateway,'enable','off');
      set(handles1.txt_usb,'string','DISCONNECTED');
      set(handles1.txt_usb,'BackgroundColor','red');
      set(handles1.conectar_gateway_slave,'enable','on');
      set(handles1.desconectar_gateway_slave,'enable','off');
      set(handles1.txt_bt,'string','DISCONNECTED');
      set(handles1.txt_bt,'BackgroundColor','red');
      errorbox('CONNECTION MASTER-GATEWAY : LOSS');
      data_time = fix(clock);
      data_time1 = num2str(data_time(1));
      data_time2 = num2str(data_time(2));
      data_time3 = num2str(data_time(3));
      data_time4 = num2str(data_time(4));
      data_time5 = num2str(data_time(5));
      data_time6 = num2str(data_time(6));
      valor = get(handles1.evento,'string');
      set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
      valor = get(handles1.evento,'string');
      set(handles1.evento,'string','CONNECTION MASTER-GATEWAY : LOSS');
      conectado = 0;

   else
      gravedad = get(handles1.menu_gravedad,'value');

      switch gravedad
         case 1 %G:+/-2g
            dato_g=1;
         case 2 %G:+/-4g
            dato_g=2;
         case 3 %G:+/-8g
            dato_g=3;
         case 4 %G:+/-16g
            dato_g=4;
      end

      SEC = 0;
```

```
ACK = 6;
CMD = 'G';
DATA = dato_g;
pkt_gravedad = char(SendtoSlave(SEC,ACK,CMD,DATA));
SEC = pkt_gravedad(3);
data_out = uint8(pkt_gravedad);
calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(7), uint8(7),
uint8(time_out_write)); % Se envia el dato al PIC
[aa,bb,data_in,dd] = calllib('libreria', 'MPUSBRead',my_in_pipe, data_in, uint8(64), uint8(64),
uint8(time_out_read)); % Se recibe el dato que envia el PIC
valid_pkt = CheckData(CMD,data_in,dd);

switch valid_pkt
  case 0 %pkt valido
    data_time = fix(clock);
    data_time1 = num2str(data_time(1));
    data_time2 = num2str(data_time(2));
    data_time3 = num2str(data_time(3));
    data_time4 = num2str(data_time(4));
    data_time5 = num2str(data_time(5));
    data_time6 = num2str(data_time(6));
    valor = get(handles1.evento,'string');
    set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
    data_g = data_in(12);
    switch data_g
      case 1
        gravedad = 1;
        msgbox('SET GRAVITY : +/-2g');
        set(handles1.txt_gravedad,'string','2g');
        valor = get(handles1.evento,'string');
        set(handles1.evento,'string',[valor,char(13),'NEW GRAVITY : +/-2g']);
      case 2
        gravedad = 2;
        msgbox('SET GRAVITY : +/-4g');
        set(handles1.txt_gravedad,'string','4g');
        valor = get(handles1.evento,'string');
        set(handles1.evento,'string',[valor,char(13),'NEW GRAVITY : +/-4g']);
      case 3
        gravedad = 3;
        msgbox('SET GRAVITY : +/-8g');
        set(handles1.txt_gravedad,'string','8g');
        valor = get(handles1.evento,'string');
        set(handles1.evento,'string',[valor,char(13),'NEW GRAVITY : +/-8g']);
      case 4
        gravedad = 4;
        msgbox('SET GRAVITY : +/-16g');
        set(handles1.txt_gravedad,'string','16g');
        valor = get(handles1.evento,'string');
        set(handles1.evento,'string',[valor,char(13),'NEW GRAVITY : +/-16g']);
    end
    %pause(2);
    start(t);
```

```
        case 1%pkt con error
          %stop(t);
          fail_gravedad = fail_gravedad+1;
          if fail_gravedad == max_error
            fail_gravedad = 0;
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'MAXIMUM ERROR RX PKT
GRAVITY']);
            msgbox('RESTART,CHECK CONNECTION GATEWAY - SLAVE');

          else
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'ERROR SETTING NEW GRAVITY']);
            errordlg('ERROR SETTING NEW GRAVITY');
            %pause(2);
            start(t);
          end

        case 2%pkt no recibido
          %stop(t);
          fail_gravedad = fail_gravedad+1;
          if fail_gravedad == max_error
            fail_gravedad = 0;
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
```

```
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'MAXIMUM TIMEOUT WAITING PKT
SETTING NEW GRAVITY']);
            msgbox('RESTART,CHECK CONNECTION GATEWAY - SLAVE LOSS');
            %stop(t);
          else
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'PKT SETTINGS NEW GRAVITY NOT
RECEIVED']);
            errordlg('PKT SETTING NEW GRAVITY NOT RECEIVED');
            %pause(2);
            start(t);
          end
      end
   end
%guidata(hObject, handles);



% --- Executes on button press in muestreo.
function muestreo_Callback(hObject, eventdata, handles)
% hObject    handle to muestreo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global conectado ACK SEC t data_in data_out my_in_pipe my_out_pipe vid_pid_norm valor
fail_sample max_error handles1 sample time_out_write time_out_read

stop(t);

   [conectado] = calllib('libreria','MPUSBGetDeviceCount',vid_pid_norm);
   if (conectado ==0)
      stop (t);
      set(handles1.conectar_master_gateway,'enable','on');
      set(handles1.desconectar_master_gateway,'enable','off');
      set(handles1.txt_usb,'string','DISCONNECTED');
      set(handles1.txt_usb,'BackgroundColor','red');
      set(handles1.conectar_gateway_slave,'enable','on');
      set(handles1.desconectar_gateway_slave,'enable','off');
      set(handles1.txt_bt,'string','DISCONNECTED');
      set(handles1.txt_bt,'BackgroundColor','red');
      errorbox('CONNECTION MASTER-GATEWAY : LOSS');
```

```
      data_time = fix(clock);
      data_time1 = num2str(data_time(1));
      data_time2 = num2str(data_time(2));
      data_time3 = num2str(data_time(3));
      data_time4 = num2str(data_time(4));
      data_time5 = num2str(data_time(5));
      data_time6 = num2str(data_time(6));
      valor = get(handles1.evento,'string');
      set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
      valor = get(handles1.evento,'string');
      set(handles1.evento,'string','CONNECTION MASTER-GATEWAY : LOSS');
      conectado = 0;
   else
      sample = get(handles1.menu_muestreo,'value');

      switch sample
         case 1 %F:12.5Hz
            dato_s=1;
         case 2 %F:25Hz
            dato_s=2;
         case 3 %F:50Hz
            dato_s=3;
         case 4 %F:100Hz
            dato_s=4;
      end

      SEC = 0
      ACK = 6;
      CMD = 'R';
      DATA = dato_s;
      pkt_muestreo = char(SendtoSlave(SEC,ACK,CMD,DATA));
      SEC = pkt_muestreo(3);
      data_out = uint8(pkt_muestreo);
      calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(7), uint8(7),
uint8(time_out_write)); % Se envia el dato al PIC
      [aa,bb,data_in,dd] = calllib('libreria', 'MPUSBRead',my_in_pipe, data_in, uint8(64), uint8(64),
uint8(time_out_read)); % Se recibe el dato que envia el PIC

      valid_pkt = CheckData(CMD,data_in,dd);

      switch valid_pkt
         case 0 %pkt valido
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
```

```
            data_s = data_in(12);
            switch data_s
              case 1
                 sample = 1;
                 msgbox('SET FRECUENCY : 12.5Hz');
                 set(handles1.txt_sample,'string','12.5Hz');
                 valor = get(handles1.evento,'string');
                 set(handles1.evento,'string',[valor,char(13),'NEW FRECUENCY : 12.5Hz']);

              case 2
                 sample = 2;
                 msgbox('SET FRECUENCY : 25Hz');
                 set(handles1.txt_sample,'string','25Hz');
                 valor = get(handles1.evento,'string');
                 set(handles1.evento,'string',[valor,char(13),'NEW FRECUENCY : 25Hz']);
              case 3
                 sample = 3;
                 msgbox('SET FRECUENCY : 50Hz');
                 set(handles1.txt_sample,'string','50Hz');
                 valor = get(handles1.evento,'string');
                 set(handles1.evento,'string',[valor,char(13),'NEW FRECUENCY : 50Hz']);
              case 4
                 sample = 4;
                 msgbox('SET FRECUENCY : 100Hz');
                 set(handles1.txt_sample,'string','100Hz');
                 valor = get(handles1.evento,'string');
                 set(handles1.evento,'string',[valor,char(13),'NEW FRECUENCY : 100Hz']);

            end
            %pause(2);
            start(t);

          case 1%pkt con error
            %stop(t);
            fail_sample = fail_sample+1;
            if fail_sample == max_error
               fail_sample = 0;
               data_time = fix(clock);
               data_time1 = num2str(data_time(1));
               data_time2 = num2str(data_time(2));
               data_time3 = num2str(data_time(3));
               data_time4 = num2str(data_time(4));
               data_time5 = num2str(data_time(5));
               data_time6 = num2str(data_time(6));
               valor = get(handles1.evento,'string');
               set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
               valor = get(handles1.evento,'string');
               set(handles1.evento,'string',[valor,char(13),'MAXIMUM ERROR RX PKT
FRECUENCY']);
               msgbox('RESTART,CHECK CONNECTION GATEWAY - SLAVE');

            else
```

```
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'ERROR SETTING NEW FRECUENCY']);
            errordlg('ERROR SETTING NEW FRECUENCY');
            %pause(2);
            start(t);
          end

        case 2%pkt no recibido
          %stop(t);
          fail_sample = fail_sample + 1;
          if fail_sample == max_error
            fail_sample = 0;
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'MAXIMUM TIMEOUT WAITING PKT
SETTING NEW FRECUENCY']);
            msgbox('RESTART,CHECK CONNECTION GATEWAY - SLAVE LOSS');

          else
            data_time = fix(clock);
            data_time1 = num2str(data_time(1));
            data_time2 = num2str(data_time(2));
            data_time3 = num2str(data_time(3));
            data_time4 = num2str(data_time(4));
            data_time5 = num2str(data_time(5));
            data_time6 = num2str(data_time(6));
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
            valor = get(handles1.evento,'string');
            set(handles1.evento,'string',[valor,char(13),'PKT SETTING NEW FRECUENCY NOT
RECEIVED']);
            errordlg('PKT SETTING NEW FRECUENCY NOT RECEIVED');
            %pause(2);
```

```
            start(t);
          end
      end
    end
%guidata(hObject, handles);

function graficar_Callback(hObject, eventdata, handles)
% hObject    handle to graficar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global handles1 m_sa axis t v1 y uaz azi sec_data

stop(t);
SF = 0.04;              % factor conv N a m/s*2
Fs = 25; Ts = 1/Fs;     %periodo de muestreo
ki =2;                  %valor inicial
sec_data = m_sa(:,1);
azi = m_sa(:,2);
N=length(azi);          %tamaño del vector de aceleracion
x=1:N;x=x';
az = azi-mean(azi);     %quitar la aceleracion estatica de la gravedad
saz=smooth(az);         %suavizar datos promedio
uaz = saz*SF;           %acc en m/s*2
v = zeros(N,1);         %vector velocidad
y = zeros(N,1);         %vector desplazamiento

switch axis
  case 1
    %graficamos la data de aceleracion
      axes(handles1.axes1);
      plot(handles1.axes1,uaz,'b.-');
      grid;

    % calcular velocidad "v" integrando aceleracion
      axes(handles1.axes2);
      for k=ki:N
        v(k)=v(k-1)+(uaz(k)+uaz(k-1))*0.5*Ts;
      end
      v1 = detrend(v);
      assignin('base','data_vel_x',v1);
      plot(handles1.axes2,v1,'b.-');
      grid;

    % calcular distancia "y" integrando velocidad
      axes(handles1.axes3);
      for k=ki:N
        y(k)=y(k-1)+(v1(k)+v1(k-1))*0.5*Ts;
      end
    % ajustar data de desplazamiento
      p = polyfit(x,y,2);
      f = polyval(p,x);
      y1 = y -f;
      assignin('base','data_desplaz_x',y1);
```

```
      plot(handles1.axes3,y1,'b.-');
      grid;
      pause(1);
      start(t);

case 2
    %graficamos la data de aceleracion
      axes(handles1.axes1);
      plot(handles1.axes1,uaz,'b.-');
      grid;

    % calcular velocidad "v" integrando aceleracion
      axes(handles1.axes2);
      for k=ki:N
        v(k)=v(k-1)+(uaz(k)+uaz(k-1))*0.5*Ts;
      end
      v1 = detrend(v);
      assignin('base','data_vel_y',v1);
      plot(handles1.axes2,v1,'b.-');
      grid;

    % calcular distancia "y" integrando velocidad
      axes(handles1.axes3);
      for k=ki:N
        y(k)=y(k-1)+(v1(k)+v1(k-1))*0.5*Ts;
      end
    % ajustar data de desplazamiento
      p = polyfit(x,y,2);
      f = polyval(p,x);
      y1 = y -f;
      assignin('base','data_desplaz_y',y1);
      plot(handles1.axes3,y1,'b.-');grid
      pause(1);
      start(t);

case 3
    %graficamos la data de aceleracion
      axes(handles1.axes1);
      plot(handles1.axes1,uaz,'b.-');
      grid;

    % calcular velocidad "v" integrando aceleracion
      axes(handles1.axes2);
      for k=ki:N
        v(k)=v(k-1)+(uaz(k)+uaz(k-1))*0.5*Ts;
      end
      v1 = detrend(v);
      assignin('base','data_vel_z',v1);
      plot(handles1.axes2,v1,'b.-');
      grid;

    % calcular distancia "y" integrando velocidad
      axes(handles1.axes3);
```

```
        for k=ki:N
          y(k)=y(k-1)+(v1(k)+v1(k-1))*0.5*Ts;
        end
      % ajustar data de desplazamiento
        p = polyfit(x,y,2);
        f = polyval(p,x);
        y1 = y -f;
        assignin('base','data_desplaz_z',y1);
        plot(handles1.axes3,y1,'b.-');
        grid;
        pause(1);
        start(t);
end


% --- Executes when selected object is changed in panel_axis.
function panel_axis_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in panel_axis
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if none was selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)
global handles1 axis

if hObject == handles1.button_x
   axis = 1;
elseif hObject == handles1.button_y
   axis = 2;
elseif hObject == handles1.button_z
   axis = 3;
else
   axis = 4;
end


% --- Executes on button press in limpiar.
function limpiar_Callback(hObject, eventdata, handles)
% hObject    handle to limpiar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global t handles1
stop(t);

axes(handles1.axes1);
cla;
axes(handles1.axes2);
cla;
axes(handles1.axes3);
cla;
pause(1);
start(t);
```

```
% --- Executes on button press in exportar.
function exportar_Callback(hObject, eventdata, handles)
% hObject    handle to exportar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global handles1 t valor axis sec_data v1 y uaz
stop(t);
T=table(sec_data,uaz,v1,y,'VariableNames',{'SECUENCIA' 'ACELERACION' 'VELOCIDAD'
'DISTANCIA'});
switch axis
   case 1
      writetable(T,'Data de desplazamiento axis_x.xls');

   case 2
      writetable(T,'Data de desplazamiento axis_y.xls');

   case 3
      writetable(T,'Data de desplazamiento axis_z.xls');
end

data_time = fix(clock);
data_time1 = num2str(data_time(1));
data_time2 = num2str(data_time(2));
data_time3 = num2str(data_time(3));
data_time4 = num2str(data_time(4));
data_time5 = num2str(data_time(5));
data_time6 = num2str(data_time(6));
valor = get(handles1.evento,'string');
set(handles1.evento,'string',[valor,char(13),data_time4,':',data_time5,':',data_time6,'-
',data_time3,'/',data_time2,'/',data_time1]);
valor = get(handles1.evento,'string');
set(handles1.evento,'string','THE DATA EXPORT WAS SUCCESSFULL');
msgbox('THE DATA EXPORT WAS SUCCESSFULL');
pause(3);
start(t);


% --- Executes on button press in exit.
function exit_Callback(hObject, eventdata, handles)
% hObject    handle to exit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global t
delete(t);
delete(gcf);
clear all;
close all;
```

FUNCION SENDTOSLAVE.m

```
function [ trama ] = SendtoSlave(secuencia,ack,comando,data)
%Funcion para ordenar el paquete a enviar
global STX SEC ACK CMD DATA CHSM ETX
%%%%%%%%%% Trama from Master to Gateway %%%%%%%%%%%%
  % PKT[1]: STX
  % PKT[2]: ACK
  % PKT[3]: SEC
  % PKT[4]: CMD
  %   T : 84 -> TEST COMUNICACION
  %   A : 65 -> DATA ACELERACION
  %   G : 71 -> MODIFICAR GRAVEDAD
  %   R : 82 -> MODIFICAR FRECUENCIA MUESTREO
  %   L : 76 -> DATA DE NIVEL DE BATERIA
  %   M : 77 -> ACC.MODO MEDICION
  %   S : 83 -> ACC.MODO STANDBY
  %   V : 86 -> REINICIAR BT
  % PKT[5]: DATA
  % PKT[6]: CHSM
  % PKT[7]: ETX
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  switch comando
    case 'A'
      STX   = 2;
      ACK   = ack;
      SEC   = secuencia;
      CMD   = 'A';
      DATA  = data;
      ETX   = 3;

    case 'G'
      STX   = 2;
      ACK   = ack;
      SEC   = secuencia;
      CMD   = 'G';
      DATA  = data;
      ETX   = 3;

    case 'M'
      STX   = 2;
      ACK   = ack;
      SEC   = secuencia;
      CMD   = 'M';
      DATA  = 0;
      ETX   = 3;

    case 'L'
      STX   = 2;
      ACK   = ack;
      SEC   = secuencia;
      CMD   = 'L';
```

```
        DATA  = 0;
        ETX   = 3;

      case 'R'
        STX   = 2;
        ACK   = ack;
        SEC   = secuencia;
        CMD   = 'R';
        DATA  = data;
        ETX   = 3;

      case 'S'
        STX   = 2;
        ACK   = ack;
        SEC   = secuencia;
        CMD   = 'S';
        DATA  = 0;
        ETX   = 3;

      case 'T'
        STX   = 2;
        ACK   = ack;
        SEC   = secuencia;
        CMD   = 'T';
        DATA  = 0;
        ETX   = 3;

      case 'V'
        STX   = 2;
        ACK   = ack;
        SEC   = secuencia;
        CMD   = 'V';
        DATA  = 0;
        ETX   = 3;
    end

    CHSM = (STX + SEC + ACK + CMD + DATA + ETX);
    trama = char(STX,ACK,SEC,CMD,DATA,CHSM,ETX);
end
```

FUNCIÓN READFROMSLAVE.m

```
function [envio, pkt_send] = ReadFromSlave( pkt_rx,seq_pkt_tx,index,error)
%Funcion para validacion del paquete de aceleración recibido

%%%%%%%% PKT from Gateway to Master %%%%%%%%%%
% PKT[1]: STX
% PKT[2]: ACK/NACK
% PKT[3]: SEQ_PKT
% PKT[4]: CMD
% PKT[5]: #DATA0
% PKT[6]: #DATA1
% PKT[7]: #DATA2
% PKT[8]: #DATA3
% PKT[9]: #DATA4
% PKT[10]: axis[i]
% PKT[11]: axis[i]
% PKT[12]: axis[i]
% PKT[13]: axis[i]
% PKT[14]: axis[i]
% PKT[15]: axis[i]
% PKT[16]: axis[i+1]
% PKT[17]: axis[i+1]
% PKT[18]: axis[i+1]
% PKT[19]: axis[i+1]
% PKT[20]: axis[i+1]
% PKT[21]: axis[i+1]
% PKT[22]: axis[i+2]
% PKT[23]: axis[i+2]
% PKT[24]: axis[i+2]
% PKT[25]: axis[i+2]
% PKT[26]: axis[i+2]
% PKT[27]: axis[i+2]
% PKT[28]: axis[i+3]
% PKT[29]: axis[i+3]
% PKT[30]: axis[i+3]
% PKT[31]: axis[i+3]
% PKT[32]: axis[i+3]
% PKT[33]: axis[i+3]
% PKT[34]: axis[i+4]
% PKT[35]: axis[i+4]
% PKT[36]: axis[i+4]
% PKT[37]: axis[i+4]
% PKT[38]: axis[i+4]
% PKT[39]: axis[i+4]
% PKT[40]: IW1-> PUNTERO DE ESCRITURA
% PKT[41]: IW2-> PUNTERO DE ESCRITURA
% PKT[42]: IR1-> PUNTERO DE LECTURA
% PKT[43]: IR2-> PUNTERO DE LECTURA
% PKT[44]: CHSM
% PKT[45]: ETX
```

```
if index == 64
   data_acc = uint8(pkt_rx);
   aux = 0;
      for i=1:43
         chsm = double(data_acc(i)) + aux;
         aux = chsm;
      end
      chsm    = chsm + double(data_acc(45));
      chsm    = mod(chsm,256);
      chsm_rx = data_acc(44);
      stx     = data_acc(1);
      etx     = data_acc(45);
      cmd     = char(data_acc(4));
      ack     = data_acc(2);
      seq_pkt = data_acc(3);
      seq_pkt_ant = uint8(seq_pkt_tx);

      if chsm == chsm_rx
         if (stx == 2 && etx == 3)
            if seq_pkt == seq_pkt_ant
               if ack == 6
                  if cmd == 'A'
                     envio = 0;%data correcta
                     seq_pkt_send = ~logical(seq_pkt);
                     seq_pkt_send = uint8(seq_pkt_send);
                     ack = 6;
                     cmd = 'A';
                     dat = 0;
                     pkt_send = (char(seq_pkt_send,ack,cmd,dat))';%pkt de respuesta
                  elseif cmd == 'S'
                     envio = 1;
                     seq_pkt_send = ~logical(seq_pkt);
                     seq_pkt_send = uint8(seq_pkt_send);
                     ack = 6;
                     cmd = 0;
                     dat = 0;
                     pkt_send = (char(seq_pkt_send,ack,cmd,dat))';%pkt de respuesta
                  end

               elseif ack == 21
                  switch cmd
                     case 'E'
                     envio = 2;%E: buffer empty
                     seq_pkt_send = ~logical(seq_pkt);
                     seq_pkt_send = uint8(seq_pkt_send);
                     ack = 6;
                     cmd = 'S';
                     dat = 0;
                     pkt_send = (char(seq_pkt_send,ack,cmd,dat))';%pkt de respuesta

                     case 'F'
                     envio = 3;%F: buffer full
                     seq_pkt_send = ~logical(seq_pkt);
```

```
                seq_pkt_send = uint8(seq_pkt_send);
                ack = 6;
                cmd = 'S';
                dat = 0;
                pkt_send = (char(seq_pkt_send,ack,cmd,dat))';%pkt de respuesta

                case 'N'
                envio = 4;%trama invalida, se descarta el pkt y se retransmite
                seq_pkt_send = seq_pkt_ant;
                ack = 6;
                cmd = 'A';
                dat = 0;
                pkt_send = (char(seq_pkt_send,ack,cmd,dat))';%pkt de respuesta
              end
            end
          else
            envio = 5;%data repetida, se descarta el pkt y se retransmite
            seq_pkt_send = seq_pkt_ant;
            ack = 6;
            cmd = 'A';
            dat = 0;
            pkt_send = (char(seq_pkt_send,ack,cmd,dat))';%pkt de respuesta
          end
        else
          envio = 6;%Pkt invalido por stx y etx incorrectos , se reenvia el comando
          seq_pkt_send = seq_pkt_ant;
          ack = 21;%NACK
          cmd = 'A';
          dat = 0;
          pkt_send = (char(seq_pkt_send,ack,cmd,dat))';%pkt de respuesta
        end
      else
        envio = 7;%Pkt invalido por error chsm, se reenvia el comando
        seq_pkt_send = seq_pkt_ant;
        ack = 21;%NACK
        cmd = 'A';
        dat = 0;
        pkt_send = (char(seq_pkt_send,ack,cmd,dat))';%pkt de respuesta
      end
else
  if error < 4
    envio = 8;%timeout,no data recibida, se reenvia el comando
    seq_pkt_send = uint8(seq_pkt_tx);
    ack = 21;%NACK
    cmd = 'A';
    dat = 0;
    pkt_send = (char(seq_pkt_send,ack,cmd,dat))';%pkt de respuesta
  else
    envio = 9;%timeout limite, fallo en la comunicacion, restart!!
    seq_pkt_send = uint8(seq_pkt_tx);
    ack = 6;
    cmd = 'S';
    dat = 0;
```

```
        pkt_send = (char(seq_pkt_send,ack,cmd,dat))';%pkt de respuesta
      end
    end
end
```

FUNCIÓN CHECKDATA.m

```
function [sms] = CheckData(cmd_send,data,index_dat)
%Validacion del paquete de nivel, stop, inicio, gravedad, muestreo, conexión y desconexión de
Slave&Gateway  recibido
%%%%%%%%% Trama from Gateway to Master %%%%%%%%%%%
% TRAMA[1]: STX
% TRAMA[2]: ACK/NACK
% TRAMA[3]: SEQ_PKT
% TRAMA[4]: CMD
% TRAMA[5]: #DATA0
% TRAMA[6]: #DATA1
% TRAMA[7]: #DATA2
% TRAMA[8]: #DATA3
% TRAMA[9]: #DATA4
% TRAMA[10]: axis[i]
% TRAMA[11]: axis[i]
% TRAMA[12]: axis[i]
% TRAMA[13]: axis[i]
% TRAMA[14]: axis[i]
% TRAMA[15]: axis[i]
% TRAMA[16]: axis[i+1]
% TRAMA[17]: axis[i+1]
% TRAMA[18]: axis[i+1]
% TRAMA[19]: axis[i+1]
% TRAMA[20]: axis[i+1]
% TRAMA[21]: axis[i+1]
% TRAMA[22]: axis[i+2]
% TRAMA[23]: axis[i+2]
% TRAMA[24]: axis[i+2]
% TRAMA[25]: axis[i+2]
% TRAMA[26]: axis[i+2]
% TRAMA[27]: axis[i+2]
% TRAMA[28]: axis[i+3]
% TRAMA[29]: axis[i+3]
% TRAMA[30]: axis[i+3]
% TRAMA[31]: axis[i+3]
% TRAMA[32]: axis[i+3]
% TRAMA[33]: axis[i+3]
% TRAMA[34]: axis[i+4]
% TRAMA[35]: axis[i+4]
% TRAMA[36]: axis[i+4]
% TRAMA[37]: axis[i+4]
% TRAMA[38]: axis[i+4]
% TRAMA[39]: axis[i+4]
```

```
% TRAMA[40]: IW1-> PUNTERO DE ESCRITURA
% TRAMA[41]: IW2-> PUNTERO DE ESCRITURA
% TRAMA[42]: IR1-> PUNTERO DE LECTURA
% TRAMA[43]: IR2-> PUNTERO DE LECTURA
% TRAMA[44]: CHSM
% TRAMA[45]: ETX
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  length_pkt = 64;%tamaño de paquete
  stx = 2;
  etx = 3;

  if  index_dat == length_pkt %comprobamos si se recibio data o expiro el timeout
    aux = 0;
    for i=1:43
      chsm = double(data(i)) + aux;
      aux = chsm;
    end
    chsm    = chsm + double(data(45));
    chsm    = mod(chsm,256);
    chsm_rx = data(44);
    stx_rx     = data(1);
    etx_rx     = data(45);
    comando = char(data(4));

    if chsm == chsm_rx
      if (stx_rx == stx && etx_rx == etx)
        if comando == cmd_send
          if comando == 'L'
            sms=0;
          else
          data1  = char(data(10));
          data2  = char(data(11));
          data= strcat(data1,data2);
          if (strcmp(data,'OK') || strcmp(data,'OA') || strcmp(data,'OT'))
            sms = 0; %pkt ok!!
          else
            sms = 1; %pkt con error!!
          end
          end
        else
           sms = 1;
        end
      else
        sms = 1;
      end
    else
      sms = 1;
    end
  else
    sms = 2; %pkt no recibido
  end
end
```

PROGRAMACIÓN EN PIC C DE LA TARJETA ADQ SLAVE

```
/********* SOFTWARE ADQ Y ENVIO DE DATA ENTRE SLAVE & GATEWAY *********/

#include <18f2550.h>
#device adc=10
#device HIGH_INTS = TRUE
#include <math.h>
#priority INT_RDA,INT_EXT2,INT_TIMER1

/************************ SELECCION DE FUSIBLES ****************************/

#fuses HS       //cristal > 4Mhz
#fuses NOWDT    //deshabilitamos el watchdog
#fuses NOLVP    //inabilitacion de programacion a bajo voltaje
#fuses NODEBUG  //depurador desactivado
#fuses CPUDIV1  //clock del CPU sin PLL
#fuses MCLR     //habilitamos el reset por hadware
#fuses NOPROTECT //fusible para no proteger el codigo contra escritura
#FUSES NOBROWNOUT //No brownout reset
#fuses PUT      // temporizador de encendido activado

/********************* DIRECTIVAS PARA COMUNICACION ************************/

#use delay (crystal = 20M, clock=20M)
#use I2C (MASTER, SDA = PIN_B0, SCL = PIN_B1, FAST=400000, FORCE_HW)
#use RS232 (BAUD=38400, BITS=8, XMIT=PIN_C6, RCV=PIN_C7, PARITY=N)

/****************** DEFINICION DE REGISTROS DEL ADXL345 ********************/

#define DATA_FORMAT 0x31  //registro para configurar resolucion,rango,formato,sensibilidad
#define POWER_CTL  0x2D  //registro para iniciar la medicion
#define ADXLW      0xA6  //registro para modo escritura
#define ADXLR      0xA7  //registro para modo lectura
#define DATA       0x32  //registro de ejes
#define BW_RATE    0x2C  //registro de ancho de banda y data rate
#define INT_SOURCE 0x30  //registro para verificar el DATA_READY
#define INT_MAP    0x2F  //registro para setear DATA_READY en INT1
#define INT_ENABLE 0x2E  //registro para habilitar INT1
#define FIFO_CTL   0x38  //registro para el status del FIFO

/******** DECLARACION DE REGISTROS DEL PIC A TRAVES DE LA RAM **********/

#BYTE TRISA = 0xF92
#BYTE TRISB = 0xF93
#BYTE TRISC = 0XF94
```

```
#BYTE TRISD = 0XF95
#BYTE PORTA = 0xF80
#BYTE PORTB = 0XF81
#BYTE PORTC = 0XF82
#BYTE PORTD = 0XF83

#BIT TRISLEVEL = 0XF94.0
#BIT TRISTICK = 0XF94.1

#BIT LEVEL = 0xF82.0
#BIT TICK = 0xF82.1

/********************** DECLARACION DE VARIABLES ***************************/

/***************** Variables y constantes para Rx de PKT Gateway&Slave ***************/
#define stx 2
#define etx 3
#define ack 6
#define nack 21
#define length_pkt_gateway 7
#define length_pkt_slave 45
#define time_min1 10
#define time_wakeup_acc1 82
#define time_wakeup_acc2 42
#define time_wakeup_acc3 22

char PktFromGateway[length_pkt_gateway]; //buffer para Rx paquete proveniente del Gateway
int8 index_pkt=0;      //variable para el indice de paquete de rx
int8 chsm_slave;       //variable para almacenar el chsm realizado por el Slave
int8 chsm_aux_slave=0; //variable para almacenar el valor temporal del chsm
int8 chsm_slave_rx;    //variable para almacenar el campo del paquete CHSM recibido
int8 stx_pkt_gateway;  //variable para iniciador de paquete
int8 etx_pkt_gateway;  //varable para terminador de paquete
int8 ack_pkt_gateway;  //variable para confirmacion de rx paquete
int8 data_pkt_gateway; //variable para almacenar el campo de data del paquete
char cmd_pkt_gateway;  //variable para alamcenar el tipo de paquete recibido
short sec_pkt_gateway; //secuenciador de paquete Gateway

/******************** Variables para Tx PK Slave -> Gateway ********************/

int8 PktToGateway[length_pkt_slave]; //buffer para Tx de data al Gateway
short sec_pkt_slave=0; //secuenciador de paquete Slave con valor inicial 0
int8 chsm_slave_tx;    //variable para almacenar el campo del paquete CHSM a enviar
int8 i,j,k,t=0;
```

```
/***** Estructura para almacenar data de aceleracion en el buffer circular ****/

#define cb_len 20 //tamaño del buffer circular

struct acc_data
{
  char ax[6];
  char ay[6];
  char az[6];
  char data_seq[5];
} ;

struct acc_data cb_data[cb_len];
int iw=0;  //index para ingresar datos al buffer circular
int ir=0;  //index para extraer datos al buffer circular
int diff_iwr=0;  //variable para almacenar la diferencia de punteros
int temp=0; //variable temporal usada en escritura del buffer circular
long dat_seq; //variable para secuencia de data de aceleracion
char pointer_iw[2]; //vector de tipo caracter para almacenar puntero de escritura
char pointer_ir[2]; //vector de tipo caracter para almacenar puntero de lectura
char diff_pointer[2];  //vector de tipo caracter para almecenar la diferencia de punteros
int8 ik,ik_1,ik_2,ik_3,ik_4;
int8 index=0;

/********** Variables para almacenar la lectura de c/eje byte por byte ***********/

signed int x[2]; //buffer para almacenar la lectura del eje LSBX y MSBX
signed int y[2]; //buffer para almacenar la lectura del eje LSBY y MSBY
signed int z[2]; //buffer para almacenar la lectura del eje LSBZ y MSBZ

signed long a_x;  //variable para almacenar el valor del eje x de dos bytes con signo
signed long a_y;  //variable para almacenar el valor del eje y de dos bytes con signo
signed long a_z;  //variable para almacenar el valor del eje z de dos bytes con signo

/*************** Variables para obtener datos de nivel de bateria **************/

#define numtovolts 0.0048875855 // valor para convertir el valor del ADC a su equivalente en voltios
#define level_low 3.75 //valor que indica un nivel de bateria bajo

long battery_num;          //variable para almacenar la lectura del modulo ADC
float battery_volts;      //variable para almacenar el dato de voltaje de la bateria
char data_level[5];   //buffer para almacenar como char de la data de nivel bateria
int g_actual;         //variable para almacenar el valor de gravedad actual
int f_actual;         //variable para almacenar el valor de la tasa de muestreo actual
long recharge_timer;
```

```
/****** INICIALIZACION I/O, MODULOS DEL uC Y REGISTROS DEL ADXL345 ********/

Void init_cpu()
 {
  /***** Configuracion del puerto C *****/

   TRISLEVEL = 0;
   TRISTICK  = 0;
   LEVEL = 0;
   TICK=0;

  /******************** Configuracion del modulo ADC ************************/

   setup_adc_ports(AN0|VSS_VDD); //seleccionamos el pin A0 como analogico
   setup_adc(ADC_CLOCK_INTERNAL|ADC_TAD_MUL_12); // Clock interno para conversion
   setup_ccp2(CCP_OFF); //modulo ccp desactivado

  /*************** Temporizador para control de TX-RX de data **************/
   setup_timer_1(T1_INTERNAL | T1_DIV_BY_8);

  /************** Configuracion del chip ADXL345 mediante bus I2C **********/
  /*** Secuencia para escribir en el chip
      1-. Start
      2-. Setear el registro para colocar chip en modo escritura
      3-. Setear el registro a configurar
      4-. Escribir el valor a cargar en el registro
      5-. Stop
  ******************************************************************************
  /*** Parametros a configurar
       Registro    : DATA_FORMAT
       Resolucion  : 13 bits
       Sensibilidad: 4mg/LSB - Full Resolution
       Rango       : +/-16g
       Interrupt   : Flanco descendente
       Formato data: Justificado a la derecha con extension de signo  */

    i2c_start ();
    i2c_write (ADXLW);
    i2c_write (DATA_FORMAT);
    i2c_write (0x0B);
    i2c_stop  ();
    g_actual = 4;
  /*****************************************************************************/
  /*** Parametros a configurar
       Registro    : BW_RATE
       Ancho Banda: 50 Hz
```

```
    Data Rate    : 100 Hz
                                                    */
  i2c_start ();
  i2c_write (ADXLW);
  i2c_write (BW_RATE);
  i2c_write (0x0A);
  i2c_stop  ();
  f_actual = 4;
/*****************************************************************************/
/*** Parametros a configurar
    Registro: POWER_CTL
    Medicion: Desactivada
                                                    */
  i2c_start ();
  i2c_write (ADXLW);
  i2c_write (POWER_CTL);
  i2c_write (0x00);
  i2c_stop  ();
/*****************************************************************************/
/*** Parametros a configurar
    Registro: FIFO_CTL
    FIFO     : OFF

                                                */
  i2c_start ();
  i2c_write (adxlw);
  i2c_write (FIFO_CTL);
  i2c_write (0x00);
  i2c_stop  ();
/*****************************************************************************/
/*** Parametros a configurar
    Registro    : INT_MAP
    DATA_READY: Seteado en la interrupcion 1(INT1) del chip
                                                    */
  i2c_start ();
  i2c_write (ADXLW);
  i2c_write (INT_MAP);
  i2c_write (0x7F);
  i2c_stop  ();
/*****************************************************************************/
/*** Parametros a configurar
    Registro     : INT_ENABLE
    Interrupcion: INT1 ON
                                                    */
  i2c_start ();
  i2c_write (ADXLW);
```

```
    i2c_write (INT_ENABLE);
    i2c_write (0x80);
    i2c_stop  ();
  /*****************************************************************************/

  /******************** Test de inicio de microcontrolador ********************/
      TICK = 1;
      delay_ms(300);
      TICK = 0;
      delay_ms(300);
      TICK = 1;
      delay_ms(300);
      TICK = 0;
  /*************************************************/

  /****************** Habilitacion de interrupciones del Uc ******************/
  /**** INTERRUPCIONES:
       INT_RDA    : Interrupcion por RS232 -> RX DATA
       INT_TIMER1: Interrupcion por desbordamiento del Timer 1
       INT_EXT2   : Interrupcion por cambio de estado en RB0 -> INT1 CHIP
       INT_GLOBAL: Interrupciones globales
                                          */
   enable_interrupts  (INT_RDA);
   ext_int_edge      (2, L_TO_H);
   disable_interrupts (INT_EXT2);
   disable_interrupts (INT_TIMER1);
   enable_interrupts  (GLOBAL);

 }

/***************** RUTINA PARA TX DATA AL GATEWAY **********************/

Void SendtoGateway(char comando,int8 dato,int8 eje)
{
/***************** PKT de Tx Slave -> Gateway ************
          PktToGateway[0]:STX
          PktToGateway[1]:ACK/NACK
          PktToGateway[2]:SECUENCIA DATA
          PktToGateway[3]:CMD
          PktToGateway[4]:#DATA[i]
          PktToGateway[5]:#DATA[i]
          PktToGateway[6]:#DATA[i]
          PktToGateway[7]:#DATA[i]
          PktToGateway[8]:#DATA[i]
          PktToGateway[9]:Data axis[i]
          PktToGateway[10]:Data axis[i]
```

```
        PktToGateway[11]:Data axis[i]
        PktToGateway[12]:Data axis[i]
        PktToGateway[13]:Data axis[i]
        PktToGateway[14]:Data axis[i]
        PktToGateway[15]:Data axis[i+1]
        PktToGateway[16]:Data axis[i+1]
        PktToGateway[17]:Data axis[i+1]
        PktToGateway[18]:Data axis[i+1]
        PktToGateway[19]:Data axis[i+1]
        PktToGateway[20]:Data axis[i+1]
        PktToGateway[21]:Data axis[i+2]
        PktToGateway[22]:Data axis[i+2]
        PktToGateway[23]:Data axis[i+2]
        PktToGateway[24]:Data axis[i+2]
        PktToGateway[25]:Data axis[i+2]
        PktToGateway[26]:Data axis[i+2]
        PktToGateway[27]=Data axis[i+3]
        PktToGateway[28]=Data axis[i+3]
        PktToGateway[29]=Data axis[i+3]
        PktToGateway[30]=Data axis[i+3]
        PktToGateway[31]=Data axis[i+3]
        PktToGateway[32]=Data axis[i+3]
        PktToGateway[33]=Data axis[i+4]
        PktToGateway[34]=Data axis[i+4]
        PktToGateway[35]=Data axis[i+4]
        PktToGateway[36]=Data axis[i+4]
        PktToGateway[37]=Data axis[i+4]
        PktToGateway[38]=Data axis[i+4]
        PktFromSlave[39]=Data pointer_iw[0]
        PktFromSlave[40]=Data pointer_iw[1]
        PktFromSlave[41]=Data pointer_ir[0]
        PktFromSlave[42]=Data pointer_ir[1]
        PktFromSlave[43]=CHSM
        PktFromSlave[44]=ETX

**************************************************************/

  switch (comando)
  {
    /** ENVIO PKT CON DATA DE ACELERACION **/
    case 'A':
    ik = dato;
    if(ik == 0)
    {
      ik_1 = cb_len-1;
      ik_2 = ik_1-1;
```

```
  ik_3 = ik_2-1;
  ik_4 = ik_3-1;

}
else
{
  ik_1 = ik-1;
  ik_2 = ik_1-1;
  ik_3 = ik_2-1;
  ik_4 = ik_3-1;

}

PktToGateway[0]=stx;
PktToGateway[1]=ack;
PktToGateway[2]=sec_pkt_slave;
PktToGateway[3]='A';
PktToGateway[4]=cb_data[ik].data_seq[0];
PktToGateway[5]=cb_data[ik].data_seq[1];
PktToGateway[6]=cb_data[ik].data_seq[2];
PktToGateway[7]=cb_data[ik].data_seq[3];
PktToGateway[8]=cb_data[ik].data_seq[4];



switch (eje)
{
  case 1:
  for (i=9;i<15;i++)
  {
    PktToGateway[i]=cb_data[ik_4].ax[index];
    index++;
  }
    index = 0;

  for (i=15;i<21;i++)
  {
    PktToGateway[i]=cb_data[ik_3].ax[index];
    index++;
  }
    index = 0;

  for (i=21;i<27;i++)
  {
    PktToGateway[i]=cb_data[ik_2].ax[index];
    index++;
```

```
}
  index = 0;

for (i=27;i<33;i++)
{
  PktToGateway[i]=cb_data[ik_1].ax[index];
  index++;
}
  index = 0;

for (i=33;i<39;i++)
{
  PktToGateway[i]=cb_data[ik].ax[index];
  index++;
}
  index = 0;
break;

case 2:
for (i=9;i<15;i++)
{
  PktToGateway[i]=cb_data[ik_4].ay[index];
  index++;
}
  index = 0;

for (i=15;i<21;i++)
{
  PktToGateway[i]=cb_data[ik_3].ay[index];
  index++;
}
  index = 0;

for (i=21;i<27;i++)
{
  PktToGateway[i]=cb_data[ik_2].ay[index];
  index++;
}
  index = 0;

for (i=27;i<33;i++)
{
  PktToGateway[i]=cb_data[ik_1].ay[index];
  index++;
}
  index = 0;
```

```
for (i=33;i<39;i++)
{
  PktToGateway[i]=cb_data[ik].ay[index];
  index++;
}
  index = 0;
break;

case 3:
for (i=9;i<15;i++)
{
  PktToGateway[i]=cb_data[ik_4].az[index];
  index++;
}
  index = 0;

for (i=15;i<21;i++)
{
  PktToGateway[i]=cb_data[ik_3].az[index];
  index++;
}
  index = 0;

for (i=21;i<27;i++)
{
  PktToGateway[i]=cb_data[ik_2].az[index];
  index++;
}
  index = 0;

for (i=27;i<33;i++)
{
  PktToGateway[i]=cb_data[ik_1].az[index];
  index++;
}
  index = 0;

for (i=33;i<39;i++)
{
  PktToGateway[i]=cb_data[ik].az[index];
  index++;
}
  index = 0;
break;
```

```
  default: break;
  index = 0;
}

sprintf(pointer_iw,"%2u",iw);
sprintf(pointer_ir,"%2u",ir);
PktToGateway[39]=pointer_iw[0];;
PktToGateway[40]=pointer_iw[1];
PktToGateway[41]=pointer_ir[0];
PktToGateway[42]=pointer_ir[1];
sec_pkt_slave = ~sec_pkt_gateway;
break;

/** ENVIO PKT POR ERROR DE BUFFER EMPTY **/
case 'E':
PktToGateway[0]=stx;
PktToGateway[1]=ack;
PktToGateway[2]=sec_pkt_slave;
PktToGateway[3]='E';
for(k=4;k<43;k++){PktToGateway[k]=0;}
sec_pkt_slave = ~sec_pkt_gateway;
break;

/** ENVIO PKT POR ERROR DE BUFFER FULL **/
case 'F':
PktToGateway[0]=stx;
PktToGateway[1]=ack;
PktToGateway[2]=sec_pkt_slave;
PktToGateway[3]='F';
for(k=4;k<43;k++){PktToGateway[k]=0;}
sec_pkt_slave = ~sec_pkt_gateway;
break;

/** ENVIO PKT DE CONFIRMACION DE NUEVO RANGO DE GRAVEDAD **/
case 'G':
PktToGateway[0]=stx;
PktToGateway[1]=ack;
PktToGateway[2]=sec_pkt_slave;
PktToGateway[3]='G';
PktToGateway[4]=0;
PktToGateway[5]=0;
PktToGateway[6]=0;
PktToGateway[7]=0;
PktToGateway[8]=0;
PktToGateway[9]='O';
PktToGateway[10]='K';
```

```
PktToGateway[11]=dato;
for(k=12;k<43;k++){PktToGateway[k]=0;}
break;

/** ENVIO PKT DE CONFIRMACION DE INICIO DE ADXL345 **/
case 'M':
disable_interrupts(INT_TIMER1);
TICK=1;
PktToGateway[0]=stx;
PktToGateway[1]=ack;
PktToGateway[2]=sec_pkt_slave;
PktToGateway[3]='M';
PktToGateway[4]=0;
PktToGateway[5]=0;
PktToGateway[6]=0;
PktToGateway[7]=0;
PktToGateway[8]=0;
PktToGateway[9]='O';
PktToGateway[10]='K';
sprintf(pointer_iw,"%2u",iw);
sprintf(pointer_ir,"%2u",ir);
PktToGateway[11]=pointer_iw[0];
PktToGateway[12]=pointer_iw[1];
PktToGateway[13]=pointer_ir[0];
PktToGateway[14]=pointer_ir[1];
for(k=15;k<43;k++){PktToGateway[k]=0;}
break;

/** ENVIO PKT CON DATA DE NIVEL DE BATERIA **/
case 'L':
PktToGateway[0]=stx;
PktToGateway[1]=ack;
PktToGateway[2]=sec_pkt_slave;
PktToGateway[3]='L';
PktToGateway[4]=0;
PktToGateway[5]=0;
PktToGateway[6]=0;
PktToGateway[7]=0;
PktToGateway[8]=0;
PktToGateway[9]=data_level[0];
PktToGateway[10]=data_level[1];
PktToGateway[11]=data_level[2];
PktToGateway[12]=data_level[3];
PktToGateway[13]=g_actual;
PktToGateway[14]=f_actual;
diff_iwr = iw - ir;
```

```
sprintf(diff_pointer,"%2u",diff_iwr);
PktToGateway[15]=diff_pointer[0];
PktToGateway[16]=diff_pointer[1];
for(k=17;k<43;k++){PktToGateway[k]=0;}
break;

/** ENVIO PKT POR ERROR DE TRAMA INVALIDA **/
case 'N':
PktToGateway[0]=stx;
PktToGateway[1]=nack;
PktToGateway[2]=sec_pkt_slave;
PktToGateway[3]='N';
for(k=4;k<43;k++){PktToGateway[k]=0;}
break;

/** ENVIO PKT DE CONFIRMACION DE NUEVA TASA DE MUESTREO **/
case 'R':
PktToGateway[0]=stx;
PktToGateway[1]=ack;
PktToGateway[2]=sec_pkt_slave;
PktToGateway[3]='R';
PktToGateway[4]=0;
PktToGateway[5]=0;
PktToGateway[6]=0;
PktToGateway[7]=0;
PktToGateway[8]=0;
PktToGateway[9]='O';
PktToGateway[10]='K';
PktToGateway[11]=dato;
for(k=12;k<43;k++){PktToGateway[k]=0;}
break;

/** ENVIO PKT DE CONFIRMACION DE STOP DEL ADXL345 **/
case 'S':
PktToGateway[0]=stx;
PktToGateway[1]=ack;
PktToGateway[2]=sec_pkt_slave;
PktToGateway[3]='S';
PktToGateway[4]=0;
PktToGateway[5]=0;
PktToGateway[6]=0;
PktToGateway[7]=0;
PktToGateway[8]=0;
PktToGateway[9]='O';
PktToGateway[10]='K';
for(k=11;k<43;k++){PktToGateway[k]=0;}
```

```
    break;

    default:break;

  }

  for(j=0;j<43;j++)
    {
      chsm_slave_tx = chsm_aux_slave + (PktToGateway[j]);
      chsm_aux_slave = chsm_slave_tx;
    }

    chsm_aux_slave = 0;
    chsm_slave_tx = chsm_slave_tx + etx;
    PktToGateway[43]=chsm_slave_tx;
    PktToGateway[44]=etx;

    for(i=0;i<45;i++){putc(PktToGateway[i]);}

}

/************** RUTINA PARA LECTURA DEL NIVEL DE BATERIA *****************/

Void Nivel()
{
  set_adc_channel(0);
  delay_us(30);
  battery_num =read_adc();
  battery_volts = battery_num * numtovolts;
  sprintf(data_level,"%1.2f",battery_volts);
  if(battery_volts <= level_low){LEVEL=1;}
  SendtoGateway('L',0,0);
}

/***************** RUTINA PARA SETEAR RANGO DE GRAVEDAD ******************/

void Gravedad(int8 g)
{
  switch(g){

  /** Gravedad = +/-2g **/
  case 1:
  i2c_start();
  i2c_write(ADXLW);
  i2c_write(DATA_FORMAT);
  i2c_write(0x08);
```

```
  i2c_stop();
  g_actual = 1;
  break;

  /** Gravedad = +/-4g **/
  case 2:
  i2c_start();
  i2c_write(ADXLW);
  i2c_write(DATA_FORMAT);
  i2c_write(0x09);
  i2c_stop();
  g_actual = 2;
  break;

  /** Gravedad = +/-8g **/
  case 3:
  i2c_start();
  i2c_write(ADXLW);
  i2c_write(DATA_FORMAT);
  i2c_write(0xA);
  i2c_stop();
  g_actual = 3;
  break;

  /** Gravedad = +/-16g **/
  case 4:
  i2c_start();
  i2c_write(ADXLW);
  i2c_write(DATA_FORMAT);
  i2c_write(0xB);
  i2c_stop();
  g_actual = 4;
  break;

  default:break;

  }

  SendtoGateway('G',g,0);
}
```

```
/****************** RUTINA PARA SETEAR TASA DE MUESTREO ********************/
void Muestreo(int8 f)
{
  switch (f){

  /** Sample Rate = 25Hz **/
  case 1:
  i2c_start();
  i2c_write(ADXLW);
  i2c_write(BW_RATE);
  i2c_write(0x07);
  i2c_stop();
  f_actual = 1;
  break;

  /** Sample Rate = 25Hz **/
  case 2:
  i2c_start();
  i2c_write(ADXLW);
  i2c_write(BW_RATE);
  i2c_write(0x08);
  i2c_stop();
  f_actual = 2;
  break;

  /** Sample Rate = 50Hz **/
  case 3:
  i2c_start();
  i2c_write(ADXLW);
  i2c_write(BW_RATE);
  i2c_write(0x09);
  i2c_stop();
  f_actual = 3;
  break;

  /** Sample Rate = 100Hz **/
  case 4:
  i2c_start();
  i2c_write(ADXLW);
  i2c_write(BW_RATE);
  i2c_write(0xA);
  i2c_stop();
  f_actual = 4;
  break;

  default: break;
```

```
  }

  SendtoGateway('R',f,0);

 }

/*************** RUTINA PARA INICIAR LA MEDIDA DEL ADXL345 **************/

void Medicion()
{
 /********************** Seteamos el ADXL345 en modo Standby ****************/
 disable_interrupts(INT_EXT2);
 i2c_start();
 i2c_write(ADXLW);
 i2c_write(POWER_CTL);
 i2c_write(0X00);
 i2c_stop();
 /******************* Seteamos el ADXL345 en modo Medicion ******************/
 i2c_start();
 i2c_write(ADXLW);
 i2c_write(POWER_CTL);
 i2c_write(0X08);
 i2c_stop();

 /***** Iniciamos los indices de lectura,escritura y secuencia de data *****/
 iw=1; //puntero de escritura igual a la posicion 1
 ir=0; //puntero de lectura a la posicion 0
 dat_seq=0; //secuencia de datos igual a 0

 switch (f_actual)
 {
   case 1:
   delay_ms(time_wakeup_acc1); //retardo para estabilizar acelerometro para Fm = 12.5hz
   recharge_timer = 10535;
   set_timer1(recharge_timer); // recargamos el timer para una interrupcion en 80ms
   break;

   case 2:
   delay_ms(time_wakeup_acc2); //retardo para estabilizar acelerometro para Fm = 25hz
   recharge_timer = 40535;
   set_timer1(recharge_timer); // recargamos el timer para una interrupcion en 40ms
   break;

   case 3:
   delay_ms(time_wakeup_acc3); //retardo para estabilizar acelerometro para Fm = 50hz
   recharge_timer = 53035;
```

```
    set_timer1(recharge_timer); // recargamos el timer para una interrupcion en 20ms
    break;

    default:break;
  }

  /****** Habilitamos interrupciones para la toma de datos ******/
  enable_interrupts(INT_EXT2);
  enable_interrupts(INT_TIMER1);

}

/*************** RUTINA PARA DETENER LA MEDIDA DEL ADXL345 **************/

void Stop()
{

  disable_interrupts(INT_EXT2); //desactivamos la interrupcion de lectura de data

  /******************** Seteamos el ADXL345 en modo Standby ***************/
  i2c_start();
  i2c_write(ADXLW);
  i2c_write(POWER_CTL);
  i2c_write(0X00);
  i2c_stop();

  SendtoGateway('S',0,0);

}

/************ RUTINA PARA ALMACENAR LA DATA DE LOS EJES ****************/

void cb_write(signed long axis_x,signed long axis_y,signed long axis_z,long d_seq)
{

  temp = iw + 1;

  if(temp >= cb_len){temp=0;}

  if(temp == ir)  //'buffer full!!'
  {
    disable_interrupts(INT_EXT2);

    /* Seteamos el ADXL345 en modo Standby*/
    i2c_start();
    i2c_write(ADXLW);
```

```
    i2c_write(POWER_CTL);
    i2c_write(0X00);
    i2c_stop();

    SendtoGateway('F',0,0);
  }
  else
  {
    sprintf(cb_data[iw].ax,"%6Ld",axis_x);
    sprintf(cb_data[iw].ay,"%6Ld",axis_y);
    sprintf(cb_data[iw].az,"%6Ld",axis_z);
    sprintf(cb_data[iw].data_seq,"%5Ld",d_seq);
    iw = temp;
  }
}

/******************** RUTINA PARA OBTENER DATA DE LOS EJES *****************/

void cb_read(int axis)
{
  if(ir >= cb_len){ir=0;}

  if(ir == iw) //'buffer empty!!'
  {
    disable_interrupts(INT_EXT2);

    /* Seteamos el ADXL345 en modo Standby*/
    i2c_start();
    i2c_write(ADXLW);
    i2c_write(POWER_CTL);
    i2c_write(0X00);
    i2c_stop();

    SendtoGateway('E',0,0);
  }
  else
  {
      SendtoGateway('A',ir,axis);
  }

}
```

```
/************** RUTINA PARA SELECCION DE DATA A ENVIAR ******************/

void Decision(char CMD,int8 DAT)
{
  switch(CMD)
   {
     case 'A':
     cb_read(DAT);
     break;

     case 'M':
     Medicion();
     break;

     case 'G':
     Gravedad(DAT);
     break;

     case 'L':
     Nivel();
     break;

     case 'R':
     Muestreo(DAT);
     break;

     case 'S':
     Stop();
     break;

     default:break;

   }
}

/*************** INTERRUPCION PARA RX DE DATA DEL GATEWAY ***************/

#INT_RDA HIGH
void Rx_data()
{


/***************** Paquete de Rx de Gateway -> Slave *****************
            PktFromGateway[0]:STX
            PktFromGateway[1]:ACK/NACK
            PktFromGateway[2]:SECUENCIA PKT
```

```
            ****************************************
            PktFromGateway[3]:CMD
            'A': Aceleracion
            'I': Inicio
            'G': Gravedad
            'R': Rango Muestreo
            'L': Nivel Bateria
            'S': Stop
            ****************************************
            PktFromGateway[4]:Data
            PktFromGateway[5]:CHSM
            PktFromGateway[6]:ETX
*********************************************************************/

    PktFromGateway[index_pkt]=getc();
    index_pkt++;

    if(index_pkt == length_pkt_gateway)
    {

      stx_pkt_gateway = (int8)(PktFromGateway[0]);
      ack_pkt_gateway = (int8)(PktFromGateway[1]);
      etx_pkt_gateway = (int8)(PktFromGateway[6]);

      //realizamos la validacion del paquete
      if(stx_pkt_gateway==stx && etx_pkt_gateway==etx)  //comprobamos la cabecera y final del
paquete
      {
        index_pkt=0;

          for(i=0;i<5;i++)    //realizamos el checksum del paquete
          {
            chsm_slave = chsm_aux_slave + ((int8)(PktFromGateway[i]));
            chsm_aux_slave = chsm_slave;
          }

          chsm_aux_slave =0;
          chsm_slave = chsm_slave + etx_pkt_gateway;
          chsm_slave_rx = (int8)(PktFromGateway[5]);

          if(chsm_slave_rx == chsm_slave)
          {
            sec_pkt_gateway = (short)(PktFromGateway[2]);
            cmd_pkt_gateway = PktFromGateway[3];
            data_pkt_gateway = PktFromGateway[4];
```

```
            if(sec_pkt_slave == sec_pkt_gateway)
            {
              if(ack_pkt_gateway == ack && cmd_pkt_gateway == 'A')
              {
                ir=ir+5; //incrementamos el puntero de lectura en 5 para leer 5 datos del buffer
                Decision('A',data_pkt_gateway);
              }
              else
                Decision(cmd_pkt_gateway,data_pkt_gateway);
            }
            else
              {
                sec_pkt_slave = sec_pkt_gateway;
                Decision(cmd_pkt_gateway,data_pkt_gateway);
              }
          }
          else {SendtoGateway('N',0,0);};
      }
      else {SendtoGateway('N',0,0);}
    }
}

/******** INTERRUPCION PARA LECTURA DE DATA CADA 40ms ******************/

#INT_EXT2
void read_axis()
{
  TICK=1;
  i2c_start ();
  i2c_write (ADXLW);
  i2c_write (DATA);

  i2c_start ();
  i2c_write (ADXLR);

  x[0] = i2c_read (1); //Lectura del LSB del eje X
  x[1] = i2c_read (1); //Lectura del MSB del eje X
  y[0] = i2c_read (1); //Lectura del LSB del eje Y
  y[1] = i2c_read (1); //Lectura del MSB del eje Y
  z[0] = i2c_read (1); //Lectura del LSB del eje Z
  z[1] = i2c_read (0); //Lectura del MSB del eje Z

  i2c_stop ();
 /* Formamos el valor de cada eje x a partir de sus valores LSB Y MSB
   funcion : x = make16(MSB,LSB)  */
```

```
  a_x = make16(x[1],x[0]);
  a_y = make16(y[1],y[0]);
  a_z = make16(z[1],z[0]);
  dat_seq++; //incrementamos en uno la secuencia de data

  cb_write(a_x,a_y,a_z,dat_seq);
  TICK=0;

}
/*** Interrupcion para confirmacion de inicio y seguridad de dato disponible ***/

#INT_TIMER1
void timer()
{
  set_timer1(recharge_timer);
  t++;

  if (t==time_min1)
  {
    SendtoGateway('M',0,0);
    t=0;
  }
}

void main ()
 {
   init_cpu();
    while(true);

    }
```

PROGRAMACIÓN EN PIC C DE LA TARJETA DE ADQ GATEWAY

```
/* SOFTWARE DEL GATEWAY PARA ADQ Y ENVIO DE DATA AL MASTER Y SLAVE */

#include <18f4550.h>

/********************** SELECCION DE FUSIBLES ****************************/

#fuses HSPLL  //cristal > 4Mhz usando el modulo PLL
#fuses NOWDT  //deshabilitamos el watchdog
#fuses NOLVP  //inabilitacion de programacion a bajo voltaje
#fuses USBDIV  //clock modulo USB a 48MHz
#fuses NODEBUG //depurador desactivado
#fuses PLL5   // Osc = 20MHz -> Fosc / 5 = 4MHz
#fuses CPUDIV1 //clock modulos procesador a 48MHz
#fuses VREGEN  //activamos el voltage de 3.3v
#fuses MCLR  //habilitamos el reset por hadware
#fuses NOPROTECT  //fusible para no proteger el codigo contra escritura
#fuses PUT       //temporizador de encendido activado

/******************** DIRECTIVAS PARA COMUNICACION **********************/

#use delay(crystal = 20M, clock=48M)
#use rs232(baud=38400,bits=8,parity=N,xmit=pin_c6,rcv=pin_c7)

/******************** CONFIGURACION DEL MODULO USB **********************/

#define USB_HID_DEVICE    FALSE          //Deshabilitamos el uso de la clase HID
#define USB_EP1_TX_ENABLE  USB_ENABLE_BULK   //Activamos EP1(EndPoint1) para IN
bulk/interrupt transfers
#define USB_EP1_RX_ENABLE  USB_ENABLE_BULK   //Activamos  EP1(EndPoint1) for OUT
bulk/interrupt transfers
#define USB_EP1_TX_SIZE   64          //Tamaño de paquete a tx por el endpoint 1 buffer
#define USB_EP1_RX_SIZE   64           //Tamaño de paquete a rx por el endpoint 1 buffer

/******************** LIBRERIAS PARA COMUNICACION USB **********************/

#include <pic18_usb.h>       //Libreria para hardware de la serie 18Fxx5x
#include <ADQ_usb_desc_scope.h>   //Libreria para descriptores y Bulk Transfers
#include <usb.c>           //Libreria para funciones del modulo USB

/********* DECLARACION DE REGISTROS DEL PIC A TRAVES DE LA RAM *********/

#BYTE TRISA  = 0x0F92
```

```
#BYTE PORTA = 0xF80

#BYTE TRISB = 0x0F93
#BYTE PORTB = 0XF81

#BYTE TRISC = 0X0F94
#BYTE PORTC = 0XF82

#BYTE TRISD = 0X0F95
#BIT TRISBT = 0X0F95.0
#BIT TRISUSB = 0X0F95.1

#BYTE PORTD = 0XF83
#BIT CONECT_BT = 0xF83.0
#BIT CONECT_USB = 0xF83.1

#BYTE ADCON0 = 0x0FC2
#BYTE ADCON1 = 0x0FC1
#BYTE CMCON = 0x0FB4

/****************************** Declaracion de Variables ************************/

#define stx 2
#define etx 3
#define ack 6
#define length_pkt_master 7
#define length_pkt_slave 45
#define length_pkt_gateway 64
#define length_pkt_bt 8

/******************* Variables para Rx PKT Slave -> Gateway *******************/

char PktFromSlave[length_pkt_slave];  //buffer para recepcionar el paquete del Slave
int8 index_pkt_slave=0;
int8 stx_pkt_rx,etx_pkt_rx;  //variable para campos de inicio y fin de paquete
int8 sec_pkt_rx;    // variable para secuencia de paquete recibido
int8 ack_pkt_rx;    // variable para el campo de confirmacion del paquete recibido
int8 chsm_pkt_rx;   //variable para almacenar el checksum del paquete recibido
int8 data_pkt_rx;   //variable para el campo de data del paquete recibido
char cmd_pkt_rx;    //variable para el campo de comando del paquete recibido
int8 chsm_Gateway;  //variable para el checksum realizado por el gateway
int8 chsm_aux_Gateway=0;  //variable para almacenar temporalmente el valor del checksum

/******************* Variables para Rx PK Master -> Gateway*********************/

int8 PktFromMaster[length_pkt_master]; //buffer para almacenar el paquete del Master
```

```
int8 chsm_test; //variable para almacenar el checksum de la conexion o desconexion de Gateway&Slave
int8 aux_bt=0;

/********************** Variables para Tx PKT Gateway -> Master ***************/

int8 PktGatewaytoMaster[length_pkt_gateway]; //buffer para envio de paquete al Master
int8 i,j,k,a,t=0;

/********************** Variables para Tx PKT Gateway -> Slave ***************/

int8 PktGatewaytoSlave[length_pkt_master]; //buffer para envio de paquete de Gateway a Slave

/* RUTINA PARA ENVIO DE COMANDOS AT PARA TEST DE CONEXION BLUETOOTH */

void Test()
{
    set_timer1(536);  // T=40ms
    enable_interrupts (INT_TIMER1);
    printf("AT+CONB4994C714AE0"); //B4994C714AE0 es la direccion MAC del modulo BLE Slave
}

/* RUTINA PARA ENVIO DE COMANDOS AT PARA REINICIAR MODULO BLUETOOTH*/

void Restart()
{
  printf("AT+RESET");
}

/********************** INICIALIZACION I/O,MODULOS DEL Uc ********************/

void init_cpu()
 {
   /*************** CONFIGURACION DE I/O Y ESTADOS INICIALES **************/

    TRISBT = 0;
    TRISUSB = 0;

    CONECT_BT  = 0;
    CONECT_USB = 0;

    /************************ INICIALIZAMOS EL MODULO USB ********************/
     usb_init();
     usb_task();
     usb_wait_for_enumeration();
```

```
/******************** Configuracion del modulo ADC ************************/

    ADCON1 = 0x0F; // desactivamos los canales analogicos
    CMCON  = 0X07; // desactivamos el modulo comparador analogico

    /* Temporizador para control de confirmacion de conexion Gateway-Slave */

    setup_timer_1(T1_INTERNAL | T1_DIV_BY_8);
    set_timer1(536);  // T=(1/48)us*4*(65535-TMR1)*8->T=0.04s

    /****************** Habilitacion de interrupciones del Uc ******************/
    /**** INTERRUPCIONES:
        INT_RDA    : Interrupcion por RS232 -> RX DATA
        INT_TIMER1 : Interrupcion por desbordamiento del Timer 1
        INT_GLOBAL : Interrupciones globales
                                                */
    enable_interrupts  (INT_RDA);
    disable_interrupts (INT_TIMER1);
    enable_interrupts  (GLOBAL);

}

void SendtoMaster(int8 seq_pkt_slave,int8 ack_pkt_slave,char type_pkt_slave,int8
checksum_slave)
{
  switch (type_pkt_slave)
  {
    case 'A':
    PktGatewaytoMaster[0]= stx;
    PktGatewaytoMaster[1]= ack_pkt_slave;
    PktGatewaytoMaster[2]= seq_pkt_slave;
    PktGatewaytoMaster[3]= 'A';
    PktGatewaytoMaster[4]= PktFromSlave[4];
    PktGatewaytoMaster[5]= PktFromSlave[5];
    PktGatewaytoMaster[6]= PktFromSlave[6];
    PktGatewaytoMaster[7]= PktFromSlave[7];
    PktGatewaytoMaster[8]= PktFromSlave[8];
    PktGatewaytoMaster[9]= PktFromSlave[9];
    PktGatewaytoMaster[10]= PktFromSlave[10];
    PktGatewaytoMaster[11]= PktFromSlave[11];
    PktGatewaytoMaster[12]= PktFromSlave[12];
    PktGatewaytoMaster[13]= PktFromSlave[13];
    PktGatewaytoMaster[14]= PktFromSlave[14];
    PktGatewaytoMaster[15]= PktFromSlave[15];
```

```
PktGatewaytoMaster[16]= PktFromSlave[16];
PktGatewaytoMaster[17]= PktFromSlave[17];
PktGatewaytoMaster[18]= PktFromSlave[18];
PktGatewaytoMaster[19]= PktFromSlave[19];
PktGatewaytoMaster[20]= PktFromSlave[20];
PktGatewaytoMaster[21]= PktFromSlave[21];
PktGatewaytoMaster[22]= PktFromSlave[22];
PktGatewaytoMaster[23]= PktFromSlave[23];
PktGatewaytoMaster[24]= PktFromSlave[24];
PktGatewaytoMaster[25]= PktFromSlave[25];
PktGatewaytoMaster[26]= PktFromSlave[26];
PktGatewaytoMaster[27]= PktFromSlave[27];
PktGatewaytoMaster[28]= PktFromSlave[28];
PktGatewaytoMaster[29]= PktFromSlave[29];
PktGatewaytoMaster[30]= PktFromSlave[30];
PktGatewaytoMaster[31]= PktFromSlave[31];
PktGatewaytoMaster[32]= PktFromSlave[32];
PktGatewaytoMaster[33]= PktFromSlave[33];
PktGatewaytoMaster[34]= PktFromSlave[34];
PktGatewaytoMaster[35]= PktFromSlave[35];
PktGatewaytoMaster[36]= PktFromSlave[36];
PktGatewaytoMaster[37]= PktFromSlave[37];
PktGatewaytoMaster[38]= PktFromSlave[38];
PktGatewaytoMaster[39]= PktFromSlave[39];
PktGatewaytoMaster[40]= PktFromSlave[40];
PktGatewaytoMaster[41]= PktFromSlave[41];
PktGatewaytoMaster[42]= PktFromSlave[42];
PktGatewaytoMaster[43]= checksum_slave;
PktGatewaytoMaster[44]= etx;
CONECT_BT=0;
break;

/** ENVIO PKT POR ERROR DE BUFFER EMPTY **/
case 'E':
PktGatewaytoMaster[0]=stx;
PktGatewaytoMaster[1]= ack_pkt_slave;
PktGatewaytoMaster[2]= seq_pkt_slave;
PktGatewaytoMaster[3]='E';
for(k=4;k<43;k++){PktGatewaytoMaster[k]=0;}
PktGatewaytoMaster[43]= checksum_slave;
PktGatewaytoMaster[44]= etx;
break;

/** ENVIO PKT POR ERROR DE BUFFER FULL **/
case 'F':
PktGatewaytoMaster[0]=STX;
```

```
PktGatewaytoMaster[1]= ack_pkt_slave;
PktGatewaytoMaster[2]= seq_pkt_slave;
PktGatewaytoMaster[3]='F';
for(k=4;k<43;k++){PktGatewaytoMaster[k]=0;}
PktGatewaytoMaster[43]= checksum_slave;
PktGatewaytoMaster[44]= etx;
break;

/** ENVIO PKT DE CONFIRMACION DE NUEVO RANGO DE GRAVEDAD **/
case 'G':
PktGatewaytoMaster[0]=STX;
PktGatewaytoMaster[1]= ack_pkt_slave;
PktGatewaytoMaster[2]= seq_pkt_slave;
PktGatewaytoMaster[3]='G';
PktGatewaytoMaster[4]=0;
PktGatewaytoMaster[5]=0;
PktGatewaytoMaster[6]=0;
PktGatewaytoMaster[7]=0;
PktGatewaytoMaster[8]=0;
PktGatewaytoMaster[9]=PktFromSlave[9];
PktGatewaytoMaster[10]=PktFromSlave[10];
PktGatewaytoMaster[11]=PktFromSlave[11];
for(k=12;k<43;k++){PktGatewaytoMaster[k]=0;}
PktGatewaytoMaster[43]= checksum_slave;
PktGatewaytoMaster[44]= etx;
break;

/** ENVIO PKT DE CONFIRMACION DE INICIO DE ADXL345 **/
case 'M':
PktGatewaytoMaster[0]=stx;
PktGatewaytoMaster[1]= ack_pkt_slave;
PktGatewaytoMaster[2]= seq_pkt_slave;
PktGatewaytoMaster[3]='M';
PktGatewaytoMaster[4]=0;
PktGatewaytoMaster[5]=0;
PktGatewaytoMaster[6]=0;
PktGatewaytoMaster[7]=0;
PktGatewaytoMaster[8]=0;
PktGatewaytoMaster[9]=PktFromSlave[9];
PktGatewaytoMaster[10]=PktFromSlave[10];
PktGatewaytoMaster[11]=PktFromSlave[11];
PktGatewaytoMaster[12]=PktFromSlave[12];
PktGatewaytoMaster[13]=PktFromSlave[13];
PktGatewaytoMaster[14]=PktFromSlave[14];
for(k=15;k<43;k++){PktGatewaytoMaster[k]=0;}
PktGatewaytoMaster[43]= checksum_slave;
```

```
PktGatewaytoMaster[44]= etx;
Break;
/** ENVIO PKT CON DATA DE NIVEL DE BATERIA **/
case 'L':
PktGatewaytoMaster[0]=stx;
PktGatewaytoMaster[1]= ack_pkt_slave;
PktGatewaytoMaster[2]= seq_pkt_slave;
PktGatewaytoMaster[3]='L';
PktGatewaytoMaster[4]=0;
PktGatewaytoMaster[5]=0;
PktGatewaytoMaster[6]=0;
PktGatewaytoMaster[7]=0;
PktGatewaytoMaster[8]=0;
PktGatewaytoMaster[9]=PktFromSlave[9];
PktGatewaytoMaster[10]=PktFromSlave[10];
PktGatewaytoMaster[11]=PktFromSlave[11];
PktGatewaytoMaster[12]=PktFromSlave[12];
PktGatewaytoMaster[13]=PktFromSlave[13];
PktGatewaytoMaster[14]=PktFromSlave[14];
PktGatewaytoMaster[15]=PktFromSlave[15];
PktGatewaytoMaster[16]=PktFromSlave[16];
for(k=17;k<43;k++){PktGatewaytoMaster[k]=0;}
PktGatewaytoMaster[43]= checksum_slave;
PktGatewaytoMaster[44]= ETX;
break;

/** ENVIO PKT POR ERROR DE TRAMA INVALIDA **/
case 'N':
PktGatewaytoMaster[0]=STX;
PktGatewaytoMaster[1]= ack_pkt_slave;
PktGatewaytoMaster[2]= seq_pkt_slave;
PktGatewaytoMaster[3]='N';
for(k=4;k<43;k++){PktGatewaytoMaster[k]=0;}
PktGatewaytoMaster[43]= checksum_slave;
PktGatewaytoMaster[44]= etx;
break;

/** ENVIO PKT DE CONFIRMACION DE NUEVA TASA DE MUESTREO **/
case 'R':
PktGatewaytoMaster[0]=STX;
PktGatewaytoMaster[1]= ack_pkt_slave;
PktGatewaytoMaster[2]= seq_pkt_slave;
PktGatewaytoMaster[3]='R';
PktGatewaytoMaster[4]=0;
PktGatewaytoMaster[5]=0;
PktGatewaytoMaster[6]=0;
```

```
PktGatewaytoMaster[7]=0;
PktGatewaytoMaster[8]=0;
PktGatewaytoMaster[9]=PktFromSlave[9];
PktGatewaytoMaster[10]=PktFromSlave[10];
PktGatewaytoMaster[11]=PktFromSlave[11];
for(k=12;k<43;k++){PktGatewaytoMaster[k]=0;}
PktGatewaytoMaster[43]= checksum_slave;
PktGatewaytoMaster[44]= ETX;
break;

/** ENVIO PKT DE CONFIRMACION DE STOP DEL ADXL345 **/
case 'S':
PktGatewaytoMaster[0]=STX;
PktGatewaytoMaster[1]= ack_pkt_slave;
PktGatewaytoMaster[2]= seq_pkt_slave;
PktGatewaytoMaster[3]='S';
PktGatewaytoMaster[4]=0;
PktGatewaytoMaster[5]=0;
PktGatewaytoMaster[6]=0;
PktGatewaytoMaster[7]=0;
PktGatewaytoMaster[8]=0;
PktGatewaytoMaster[9]=PktFromSlave[9];
PktGatewaytoMaster[10]=PktFromSlave[10];
for(k=11;k<43;k++){PktGatewaytoMaster[k]=0;}
PktGatewaytoMaster[43]= checksum_slave;
PktGatewaytoMaster[44]= ETX;
break;

case 'T':
PktGatewaytoMaster[0]= stx;
PktGatewaytoMaster[1]= ack;
PktGatewaytoMaster[2]= sec_pkt_rx;
PktGatewaytoMaster[3]= 'T';
PktGatewaytoMaster[4]= 0;
PktGatewaytoMaster[5]= 0;
PktGatewaytoMaster[6]= 0;
PktGatewaytoMaster[7]= 0;
PktGatewaytoMaster[8]= 0;
PktGatewaytoMaster[9]= PktFromSlave[0];
PktGatewaytoMaster[10]=PktFromSlave[7];

for(k=11;k<43;k++){PktGatewaytoMaster[k]=0;}
for(a=0;a<11;a++)
{
  chsm_test = aux_bt + PktGatewaytoMaster[a];
  aux_bt = chsm_test;
```

```
    }
    chsm_test = aux_bt + etx;
    aux_bt = 0;
    PktGatewaytoMaster[43]= chsm_test;
    PktGatewaytoMaster[44]= etx;
    if( PktFromSlave[7] == 'A'){CONECT_BT=1;}
    break;

    case 'V':
    PktGatewaytoMaster[0]= stx;
    PktGatewaytoMaster[1]= ack;
    PktGatewaytoMaster[2]= sec_pkt_rx;
    PktGatewaytoMaster[3]= 'V';
    PktGatewaytoMaster[4]= 0;
    PktGatewaytoMaster[5]= 0;
    PktGatewaytoMaster[6]= 0;
    PktGatewaytoMaster[7]= 0;
    PktGatewaytoMaster[8]= 0;
    PktGatewaytoMaster[9]=PktFromSlave[0];
    PktGatewaytoMaster[10]=PktFromSlave[7];

    for(k=11;k<43;k++){PktGatewaytoMaster[k]=0;}
    for(a=0;a<11;a++)
    {
      chsm_test = aux_bt + PktGatewaytoMaster[a];
      aux_bt = chsm_test;
    }
    chsm_test = aux_bt + etx;
    aux_bt = 0;
    PktGatewaytoMaster[43]= chsm_test;
    PktGatewaytoMaster[44]= etx;
    if(PktFromSlave[7] == 'T'){CONECT_BT=0;}
    break;

    default: break;
  }

  for(k=45;k<65;k++){PktGatewaytoMaster[k]=0;}
  usb_put_packet(1, PktGatewaytoMaster, 64 , USB_DTS_TOGGLE); //enviamos el paquete de
tamaño 64bytes del EP1 al PC
}
```

```
/************** RUTINA DE ENVIO PARA TX DE DATA AL MASTER ***************/

void SendtoSlave(int8 seq_pkt_master,int8 ack_pkt_master,char type_pkt_master,int8
data_master,int8 checksum_master)
{

 /************* Pkt tx from Gateway to Slave **************
          PktGatewaytoSlave[0]= STX
          PktGatewaytoSlave[1]= ACK/NACK
          PktGatewaytoSlave[2] = SECUENCIA PKT
          ************************
          PktGatewaytoSlave[3]= CMD
          'A': Aceleracion
          'G': Gravedad
          'R': Rango Muestreo
          'L': Nivel Bateria
          'S': Stop
          'T': Test
          ************************
          PktGatewaytoSlave[4]= DATA
          PktGatewaytoSlave[5]= CHSM
          PktGatewaytoSlave[6]= EXT
 ***********************************************************/

 switch (type_pkt_master){
   case 'A':
   PktGatewaytoSlave[0]=stx;
   PktGatewaytoSlave[1]=ack_pkt_master;
   PktGatewaytoSlave[2]=seq_pkt_master;
   PktGatewaytoSlave[3]='A';
   PktGatewaytoSlave[4]=data_master;
   PktGatewaytoSlave[5]=checksum_master;
   PktGatewaytoSlave[6]=etx;
   break;

   case 'G':
   PktGatewaytoSlave[0]=stx;
   PktGatewaytoSlave[1]=ack_pkt_master;
   PktGatewaytoSlave[2]=seq_pkt_master;
   PktGatewaytoSlave[3]='G';
   PktGatewaytoSlave[4]=data_master;
   PktGatewaytoSlave[5]=checksum_master;
   PktGatewaytoSlave[6]=etx;
   break;
```

```
  case 'M':
  PktGatewaytoSlave[0]=stx;
  PktGatewaytoSlave[1]=ack_pkt_master;
  PktGatewaytoSlave[2]=seq_pkt_master;
  PktGatewaytoSlave[3]='M';
  PktGatewaytoSlave[4]=data_master;
  PktGatewaytoSlave[5]=checksum_master;
  PktGatewaytoSlave[6]=etx;
  break;

  case 'L':
  PktGatewaytoSlave[0]=stx;
  PktGatewaytoSlave[1]=ack_pkt_master;
  PktGatewaytoSlave[2]=seq_pkt_master;
  PktGatewaytoSlave[3]='L';
  PktGatewaytoSlave[4]=data_master;
  PktGatewaytoSlave[5]=checksum_master;
  PktGatewaytoSlave[6]=etx;
  break;

  case 'R':
  PktGatewaytoSlave[0]=stx;
  PktGatewaytoSlave[1]=ack_pkt_master;
  PktGatewaytoSlave[2]=seq_pkt_master;
  PktGatewaytoSlave[3]='R';
  PktGatewaytoSlave[4]=data_master;
  PktGatewaytoSlave[5]=checksum_master;
  PktGatewaytoSlave[6]=etx;
  break;

  case 'S':
  PktGatewaytoSlave[0]=stx;
  PktGatewaytoSlave[1]=ack_pkt_master;
  PktGatewaytoSlave[2]=seq_pkt_master;
  PktGatewaytoSlave[3]='S';
  PktGatewaytoSlave[4]=data_master;
  PktGatewaytoSlave[5]=checksum_master;
  PktGatewaytoSlave[6]=etx;
  break;

  default:break;

}

for(i=0;i<7;i++){putc(PktGatewaytoSlave[i]);}
```

```
}
#INT_TIMER1
void ConectGateway_Slave()
{
  set_timer1(536);  //recargamos el timer
  t++;  //incrementamos t en uno
  /*Si t = 250 -> time_wait = 10s para recibir confirmacion de conexion */
  if(t == 250)
  {
    disable_interrupts(INT_TIMER1);
    SendtoMaster(0,0,'T',0);
    t=0;
  }
}

#INT_RDA
void RxDataFromSlave()
{
  /*PKT de Rx -> ****************************************
            PktFromSlave[0]:STX
            PktFromSlave[1]:ACK/NACK
            PktFromSlave[2]:SECUENCIA DATA
            PktFromSlave[3]:CMD
              'A': Aceleracion
              'E': Buffer Empty
              'F': Buffer Full
              'G': Gravedad
              'I': Inicio
              'L': Nivel Bateria
              'N': Trama invalida
              'R': Rango Muestreo
              'S': Stop
            PktFromSlave[4]:#DATA[i]
            PktFromSlave[5]:#DATA[i]
            PktFromSlave[6]:#DATA[i]
            PktFromSlave[7]:#DATA[i]
            PktFromSlave[8]:#DATA[i]
            PktFromSlave[9]:Data axis[i]
            PktFromSlave[10]:Data axis[i]
            PktFromSlave[11]:Data axis[i]
            PktFromSlave[12]:Data axis[i]
            PktFromSlave[13]:Data axis[i]
            PktFromSlave[14]:Data axis[i]
            PktFromSlave[15]:Data axis[i+1]
            PktFromSlave[16]:Data axis[i+1]
            PktFromSlave[17]:Data axis[i+1]
```

```
PktFromSlave[18]:Data axis[i+1]
PktFromSlave[19]:Data axis[i+1]
PktFromSlave[20]:Data axis[i+1]
PktFromSlave[21]:Data axis[i+2]
PktFromSlave[22]:Data axis[i+2]
PktFromSlave[23]:Data axis[i+2]
PktFromSlave[24]:Data axis[i+2]
PktFromSlave[25]:Data axis[i+2]
PktFromSlave[26]:Data axis[i+2]
PktFromSlave[27]=Data axis[i+3]
PktFromSlave[28]=Data axis[i+3]
PktFromSlave[29]=Data axis[i+3]
PktFromSlave[30]=Data axis[i+3]
PktFromSlave[31]=Data axis[i+3]
PktFromSlave[32]=Data axis[i+3]
PktFromSlave[33]=Data axis[i+4]
PktFromSlave[34]=Data axis[i+4]
PktFromSlave[35]=Data axis[i+4]
PktFromSlave[36]=Data axis[i+4]
PktFromSlave[37]=Data axis[i+4]
PktFromSlave[38]=Data axis[i+4]
PktFromSlave[39]=Data pointer_iw[0]
PktFromSlave[40]=Data pointer_iw[1]
PktFromSlave[41]=Data pointer_ir[0]
PktFromSlave[42]=Data pointer_ir[1]
PktFromSlave[43]=CHSM
PktFromSlave[44]=ETX
```

Nota: El paquete de respuesta que se obtiene del test de comunicacion es diferente ya que el BLE Slave responde despues de 10s con el comando AT : OK+CONNA, si la conexion es exitosa o con OK+CONNF si hay un fallo al intentar conectarse.

```
**********************************************************************************/

  PktFromSlave[index_pkt_slave] = getc();
  index_pkt_slave++;

 /* Comprobamos si es un paquete de respuesta correspondiente a una conexion exitosa o fallida */

if(PktFromSlave[0]=='O'&&(PktFromSlave[7]=='A'||PktFromSlave[7]=='F')&&  index_pkt_slave
      == (length_pkt_bt))
 {
  index_pkt_slave = 0;
 }
```

```
/* Comprobamos si es un paquete de respuesta correspondiente a un reset : OK+RESET */
  Else if(PktFromSlave[0]=='O' && PktFromSlave[7]=='T' && index_pkt_slave ==
(length_pkt_bt))
  {
    index_pkt_slave = 0;
    SendtoMaster(0,0,'V',0);
  }

  /* Validamos el paquete de recepcion proveniente del Slave */
  else if(index_pkt_slave == length_pkt_slave)
  {
    stx_pkt_rx = (int8)PktFromSlave[0];
    ack_pkt_rx = (int8)PktFromSlave[1];
    etx_pkt_rx = (int8)PktFromSlave[44];
    index_pkt_slave=0;

    if(stx_pkt_rx == stx && etx_pkt_rx == etx) //comprobamos el inicio y final del paquete
    {
      for(i=0;i<length_pkt_slave-2;i++) //realizamos el checksum del paquete
      {
        chsm_Gateway = chsm_aux_Gateway + PktFromSlave[i];
        chsm_aux_Gateway = chsm_Gateway;
      }

      chsm_aux_Gateway = 0;
      chsm_Gateway = chsm_Gateway + etx_pkt_rx;
      chsm_pkt_rx = (int8)(PktFromSlave[43]);
      sec_pkt_rx = (int8)PktFromSlave[2];
      cmd_pkt_rx = PktFromSlave[3];

      if(chsm_pkt_rx == chsm_Gateway)
      {
        switch(cmd_pkt_rx)
        {
          case 'A':
          SendtoMaster(sec_pkt_rx,ack_pkt_rx,'A',chsm_pkt_rx);
          break;

          case 'E':
          SendtoMaster(sec_pkt_rx,ack_pkt_rx,'E',chsm_pkt_rx);
          break;

          case 'F':
          SendtoMaster(sec_pkt_rx,ack_pkt_rx,'F',chsm_pkt_rx);
          break;
```

```
            case 'G':
            SendtoMaster(sec_pkt_rx,ack_pkt_rx,'G',chsm_pkt_rx);
            break;

            case 'M':
            SendtoMaster(sec_pkt_rx,ack_pkt_rx,'M',chsm_pkt_rx);
            break;

            case 'L':
            SendtoMaster(sec_pkt_rx,ack_pkt_rx,'L',chsm_pkt_rx);
            break;

            case 'N':
            SendtoMaster(sec_pkt_rx,ack_pkt_rx,'N',chsm_pkt_rx);
            break;

            case 'R':
            SendtoMaster(sec_pkt_rx,ack_pkt_rx,'R',chsm_pkt_rx);
            break;

            case 'S':
            SendtoMaster(sec_pkt_rx,ack_pkt_rx,'S',chsm_pkt_rx);
            break;

            default:break;
          }
        }
      }
    }
}

void main()
{
  init_cpu();

  while(true)
  {
    usb_task();

    if(usb_enumerated())  // Si el HOST enumera al pic -> TRUE
    {
      CONECT_USB =1;

      if(usb_kbhit(1)) // Si hay data disponible en el endpoint de salida del HOST -> TRUE
      {
```

```
/******* Pkt Rx from Master ***********
   PktFromMaster[0]= STX
   PktFromMaster[1]= ACK/NACK
   PktFromMaster[2] = SECUENCIA PKT
   ************************
   PktFromMaster[3]= CMD
   'A': Aceleracion
   'G': Gravedad
   'R': Rango Muestreo
   'L': Nivel Bateria
   'M': Acc. modo medicion
   'S': Acc. modo standby
   'T': Test
   ************************
   PktFromMaster[4]= DATA
   PktFromMaster[5]= CHSM
   PktFromMaster[6]= EXT */

usb_get_packet(1,PktFromMaster,length_pkt_master);  //Recepcionamos el paquete de 7
bytes del host del EP1 y almacenamos en TramaFromMaster

stx_pkt_rx  = PktFromMaster[0];
ack_pkt_rx  = PktFromMaster[1];
sec_pkt_rx  = PktFromMaster[2];
cmd_pkt_rx  = (char)(PktFromMaster[3]);
data_pkt_rx = PktFromMaster[4];
chsm_pkt_rx = PktFromMaster[5];
etx_pkt_rx  = PktFromMaster[6];

if(stx_pkt_rx == stx && etx_pkt_rx == etx)
{
  for(i=0;i<5;i++)
  {
    chsm_Gateway = chsm_aux_Gateway + PktFromMaster[i];
    chsm_aux_Gateway = chsm_Gateway;
  }

  chsm_aux_Gateway = 0;
  chsm_Gateway = chsm_Gateway + etx_pkt_rx;

  if(chsm_Gateway == chsm_pkt_rx)
  {
     switch(cmd_pkt_rx)
     {
```

```
                case 'A':
                CONECT_BT=1;

                SendtoSlave(sec_pkt_rx,ack_pkt_rx,'A',data_pkt_rx,chsm_pkt_rx);
                break;

                case 'G':
                SendtoSlave(sec_pkt_rx,ack_pkt_rx,'G',data_pkt_rx,chsm_pkt_rx);
                break;

                 case 'L':
                SendtoSlave(sec_pkt_rx,ack_pkt_rx,'L',0,chsm_pkt_rx);
                break;

                case 'M':
                SendtoSlave(sec_pkt_rx,ack_pkt_rx,'M',0,chsm_pkt_rx);
                break;

                case 'R':
                SendtoSlave(sec_pkt_rx,ack_pkt_rx,'R',data_pkt_rx,chsm_pkt_rx);
                break;

                case 'S':
                SendtoSlave(sec_pkt_rx,ack_pkt_rx,'S',0,chsm_pkt_rx);
                break;

                case 'T':
                Test();
                break;

                case 'V':
                Restart();
                break;

                default:break;
            }
          }
        }
      }
    }
  }
}
```

3-Axis, ±2 g/±4 g/±8 g/±16 g Digital Accelerometer

ADXL345

ATURES

ralow power: as low as 40 µA in measurement mode and
).1 µA in standby mode at V_S = 2.5 V (typical)
wer consumption scales automatically with bandwidth
er-selectable resolution
ixed 10-bit resolution
ull resolution, where resolution increases with *g* range, up to 13-bit resolution at ±16 *g* (maintaining 4 mg/LSB scale factor in all *g* ranges)
bedded, patent pending FIFO technology minimizes host processor load
)/double tap detection
tivity/inactivity monitoring
e-fall detection
pply voltage range: 2.0 V to 3.6 V
voltage range: 1.7 V to V_S
I (3- and 4-wire) and I²C digital interfaces
xible interrupt modes mappable to either interrupt pin
asurement ranges selectable via serial command
ndwidth selectable via serial command
de temperature range (−40°C to +85°C)
000 *g* shock survival
free/RoHS compliant
all and thin: 3 mm × 5 mm × 1 mm LGA package

PLICATIONS

ndsets
dical instrumentation
ming and pointing devices
lustrial instrumentation
rsonal navigation devices
rd disk drive (HDD) protection
ness equipment

GENERAL DESCRIPTION

The ADXL345 is a small, thin, low power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ±16 *g*. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I²C digital interface.

The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (4 mg/LSB) enables measurement of inclination changes less than 1.0°.

Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion and if the acceleration on any axis exceeds a user-set level. Tap sensing detects single and double taps. Free-fall sensing detects if the device is falling. These functions can be mapped to one of two interrupt output pins. An integrated, patent pending 32-level first in, first out (FIFO) buffer can be used to store data to minimize host processor intervention.

Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation.

The ADXL345 is supplied in a small, thin, 3 mm × 5 mm × 1 mm, 14-lead, plastic package.

FUNCTIONAL BLOCK DIAGRAM

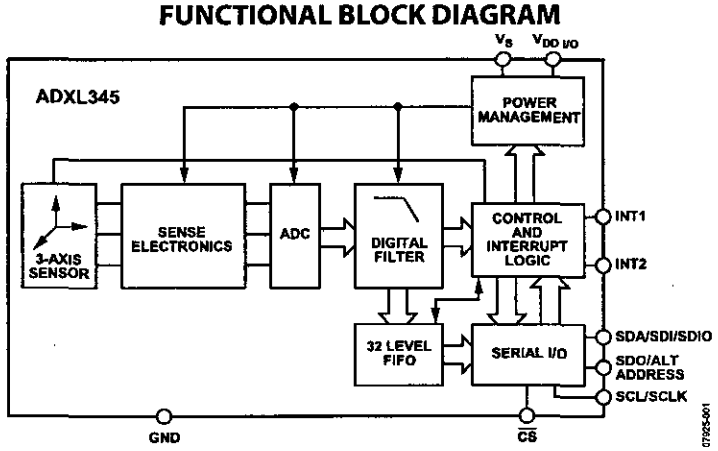


Figure 1.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A
Tel: 781.329.4700 www.analog.com
Fax: 781.461.3113

E OF CONTENTS

ION HISTORY

Revision 0: Initial Version

PECIFICATIONS

= 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V, acceleration = 0 *g*, C_S = 1 µF tantalum, C_{IO} = 0.1 µF, unless otherwise noted.

ble 1. Specifications[1]

ameter	Test Conditions	Min	Typ	Max	Unit
NSOR INPUT	Each axis				
Measurement Range	User selectable		±2, ±4, ±8, ±16		*g*
Nonlinearity	Percentage of full scale		±0.5		%
Inter-Axis Alignment Error			±0.1		Degrees
Cross-Axis Sensitivity[2]			±1		%
TPUT RESOLUTION	Each axis				
All *g* Ranges	10-bit resolution		10		Bits
±2 *g* Range	Full resolution		10		Bits
±4 *g* Range	Full resolution		11		Bits
±8 *g* Range	Full resolution		12		Bits
±16 *g* Range	Full resolution		13		Bits
NSITIVITY	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	±2 *g*, 10-bit or full resolution	232	256	286	LSB/*g*
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	±2 *g*, 10-bit or full resolution	3.5	3.9	4.3	mg/LSB
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	±4 *g*, 10-bit resolution	116	128	143	LSB/*g*
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	±4 *g*, 10-bit resolution	7.0	7.8	8.6	mg/LSB
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	±8 *g*, 10-bit resolution	58	64	71	LSB/*g*
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	±8 *g*, 10-bit resolution	14.0	15.6	17.2	mg/LSB
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	±16 *g*, 10-bit resolution	29	32	36	LSB/*g*
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	±16 *g*, 10-bit resolution	28.1	31.2	34.3	mg/LSB
Sensitivity Change Due to Temperature			±0.01		%/°C
BIAS LEVEL	Each axis				
0 *g* Output for X_{OUT}, Y_{OUT}		−150	±40	+150	mg
0 *g* Output for Z_{OUT}		−250	±80	+250	mg
0 *g* Offset vs. Temperature for x-, y-Axes			±0.8		mg/°C
0 *g* Offset vs. Temperature for z-Axis			±4.5		mg/°C
ISE PERFORMANCE					
Noise (x-, y-Axes)	Data rate = 100 Hz for ±2 *g*, 10-bit or full resolution		<1.0		LSB rms
Noise (z-Axis)	Data rate = 100 Hz for ±2 *g*, 10-bit or full resolution		<1.5		LSB rms
TPUT DATA RATE AND BANDWIDTH	User selectable				
Measurement Rate[3]		6.25		3200	Hz
F-TEST[4]	Data rate ≥ 100 Hz, 2.0 V ≤ V_S ≤ 3.6 V				
Output Change in x-Axis		0.20		2.10	*g*
Output Change in y-Axis		−2.10		−0.20	*g*
Output Change in z-Axis		0.30		3.40	*g*
WER SUPPLY					
Operating Voltage Range (V_S)		2.0	2.5	3.6	V
Interface Voltage Range ($V_{DD\ I/O}$)	V_S ≤ 2.5 V	1.7	1.8	V_S	V
	V_S ≥ 2.5 V	2.0	2.5	V_S	V
Supply Current	Data rate > 100 Hz		145		µA
	Data rate < 10 Hz		40		µA
Standby Mode Leakage Current			0.1	2	µA
Turn-On Time[5]	Data rate = 3200 Hz		1.4		ms
MPERATURE					
Operating Temperature Range		−40		+85	°C
IGHT					
Device Weight			20		mg

l minimum and maximum specifications are guaranteed. Typical specifications are not guaranteed.
oss-axis sensitivity is defined as coupling between any two axes.
ndwidth is half the output data rate.
lf-test change is defined as the output (*g*) when the SELF_TEST bit = 1 (in the DATA_FORMAT register) minus the output (*g*) when the SELF_TEST bit = 0 (in the ATA_FORMAT register). Due to device filtering, the output reaches its final value after 4 × τ when enabling or disabling self-test, where τ = 1/(data rate).
rn-on and wake-up times are determined by the user-defined bandwidth. At a 100 Hz data rate, the turn-on and wake-up times are each approximately 11.1 ms. Fo her data rates, the turn-on and wake-up times are each approximately τ + 1.1 in milliseconds, where τ = 1/(data rate).

)LUTE MAXIMUM RATINGS

ːter	Rating
ation	
\xis, Unpowered	10,000 *g*
\xis, Powered	10,000 *g*
	−0.3 V to +3.6 V
	−0.3 V to +3.6 V
?ins	−0.3 V to $V_{DD\ I/O}$ + 0.3 V or 3.6 V, whichever is less
ːr Pins	−0.3 V to +3.6 V
Short-Circuit Duration Pin to Ground)	Indefinite
ature Range	
ːred	−40°C to +105°C
ge	−40°C to +105°C

Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions above those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

THERMAL RESISTANCE

Table 3. Package Characteristics

Package Type	θ_{JA}	θ_{JC}	Device Weight
14-Terminal LGA	150°C/W	85°C/W	20 mg

ESD CAUTION

ESD (electrostatic discharge) sensitive device. Charged devices and circuit boards can discharge without detection. Although this product features patented or proprietary protection circuitry, damage may occur on devices subjected to high energy ESD. Therefore, proper ESD precautions should be taken to avoid performance degradation or loss of functionality.

IN CONFIGURATION AND FUNCTION DESCRIPTIONS

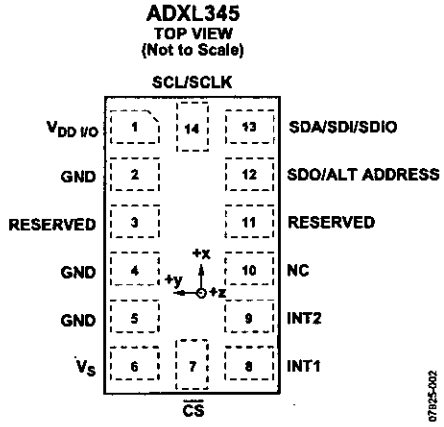


Figure 2. Pin Configuration

ble 4. Pin Function Descriptions

ı No.	Mnemonic	Description
	$V_{DD I/O}$	Digital Interface Supply Voltage.
	GND	Must be connected to ground.
	Reserved	Reserved. This pin must be connected to V_S or left open.
	GND	Must be connected to ground.
	GND	Must be connected to ground.
	V_S	Supply Voltage.
	$\overline{CS}$	Chip Select.
	INT1	Interrupt 1 Output.
	INT2	Interrupt 2 Output.
	NC	Not Internally Connected.
	Reserved	Reserved. This pin must be connected to ground or left open.
	SDO/ALT ADDRESS	Serial Data Output/Alternate I^2C Address Select.
	SDA/SDI/SDIO	Serial Data (I^2C)/Serial Data Input (SPI 4-Wire)/Serial Data Input and Output (SPI 3-Wire).
	SCL/SCLK	Serial Communications Clock.

)RY OF OPERATION

)XL345 is a complete 3-axis acceleration measurement with a selectable measurement range of ±2 *g*, ±4 *g*, ±8 *g*, g. It measures both dynamic acceleration resulting from or shock and static acceleration, such as gravity, which the device to be used as a tilt sensor.

ısor is a polysilicon surface-micromachined structure ı top of a silicon wafer. Polysilicon springs suspend the re over the surface of the wafer and provide a resistance acceleration forces.

ion of the structure is measured using differential capacitors ısist of independent fixed plates and plates attached to the mass. Acceleration deflects the beam and unbalances the tial capacitor, resulting in a sensor output whose amplitude ortional to acceleration. Phase-sensitive demodulation is determine the magnitude and polarity of the acceleration.

ER SEQUENCING

can be applied to V_S or $V_{DD\ I/O}$ in any sequence without ing the ADXL345. All possible power-on modes are ırized in Table 5. The interface voltage level is set with rface supply voltage, $V_{DD\ I/O}$, which must be present to that the ADXL345 does not create a conflict on the ınication bus. For single-supply operation, $V_{DD\ I/O}$ can be ıe as the main supply, V_S. In a dual-supply application, r, $V_{DD\ I/O}$ can differ from V_S to accommodate the desired e voltage, as long as V_S is greater than $V_{DD\ I/O}$.

s is applied, the device enters standby mode, where power ıption is minimized and the device waits for $V_{DD\ I/O}$ to be l and for the command to enter measurement mode to be d. (This command can be initiated by setting the measure ıe POWER_CTL register (Address 0x2D).) In addition, any can be written to or read from to configure the part while ice is in standby mode. It is recommended to configure the in standby mode and then to enable measurement mode. g the measure bit returns the device to the standby mode.

5. Power Sequencing

on	V_S	$V_{DD\ I/O}$	Description
)ff	Off	Off	The device is completely off, but there is a potential for a communication bus conflict.
ıbled	On	Off	The device is on in standby mode, but communication is unavailable and will create a conflict on the communication bus. The duration of this state should be minimized during power-up to prevent a conflict.
bled	Off	On	No functions are available, but the device will not create a conflict on the communication bus.
' or urement	On	On	At power-up, the device is in standby mode, awaiting a command to enter measurement mode, and all sensor functions are off. After the device is instructed to enter measurement mode, all sensor functions are available.

POWER SAVINGS

Power Modes

The ADXL345 automatically modulates its power consumption in proportion to its output data rate, as outlined in Table 6. If additional power savings is desired, a lower power mode is available. In this mode, the internal sampling rate is reduced, allowing for power savings in the 12.5 Hz to 400 Hz data rate range but at the expense of slightly greater noise. To enter lower power mode, set the LOW_POWER bit (Bit 4) in the BW_RATE register (Address 0x2C). The current consumption in low power mode is shown in Table 7 for cases where there is an advantage for using low power mode. The current consumption values shown in Table 6 and Table 7 are for a V_S of 2.5 V. Current scales linearly with V_S.

Table 6. Current Consumption vs. Data Rate
(T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Output Data Rate (Hz)	Bandwidth (Hz)	Rate Code	I_{DD} (μA)
3200	1600	1111	145
1600	800	1110	100
800	400	1101	145
400	200	1100	145
200	100	1011	145
100	50	1010	145
50	25	1001	100
25	12.5	1000	65
12.5	6.25	0111	55
6.25	3.125	0110	40

Table 7. Current Consumption vs. Data Rate, Low Power Mode
(T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Output Data Rate (Hz)	Bandwidth (Hz)	Rate Code	I_{DD} (μA)
400	200	1100	100
200	100	1011	65
100	50	1010	55
50	25	1001	50
25	12.5	1000	40
12.5	6.25	0111	40

ıto Sleep Mode

ldditional power can be saved if the ADXL345 automatically itches to sleep mode during periods of inactivity. To enable s feature, set the THRESH_INACT register (Address 0x25) d the TIME_INACT register (Address 0x26) each to a value ıt signifies inactivity (the appropriate value depends on the ɔlication), and then set the AUTO_SLEEP bit and the link bit in : POWER_CTL register (Address 0x2D). Current consumption the sub-8 Hz data rates used in this mode is typically 40 μA · a V_S of 2.5 V.

Standby Mode

For even lower power operation, standby mode can be used. In standby mode, current consumption is reduced to 0.1 μA (typical). In this mode, no measurements are made. Standby mode is entere by clearing the measure bit (Bit 3) in the POWER_CTL registeı (Address 0x2D). Placing the device into standby mode preserves the contents of FIFO.

IAL COMMUNICATIONS

d SPI digital communications are available. In both cases,)XL345 operates as a slave. I²C mode is enabled if the $\overline{CS}$ pin high to $V_{DD\ I/O}$. The $\overline{CS}$ pin should always be tied high to or be driven by an external controller because there is no t mode if the $\overline{CS}$ pin is left unconnected. Therefore, not these precautions may result in an inability to communicate ıe part. In SPI mode, the $\overline{CS}$ pin is controlled by the bus . In both SPI and I²C modes of operation, data transmitted ıe ADXL345 to the master device should be ignored during to the ADXL345.

I, either 3- or 4-wire configuration is possible, as shown connection diagrams in Figure 3 and Figure 4. Clearing bit in the DATA_FORMAT register (Address 0x31) selects mode, whereas setting the SPI bit selects 3-wire mode. aximum SPI clock speed is 5 MHz with 100 pF maximum ;, and the timing scheme follows clock polarity (CPOL) = 1 ock phase (CPHA) = 1.

ıe serial port enable line and is controlled by the SPI master. ne must go low at the start of a transmission and high at d of a transmission, as shown in Figure 5. SCLK is the ort clock and is supplied by the SPI master. It is stopped hen $\overline{CS}$ is high during a period of no transmission. SDI O are the serial data input and output, respectively. Data be sampled at the rising edge of SCLK.

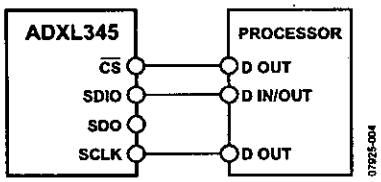


Figure 3. 3-Wire SPI Connection Diagram

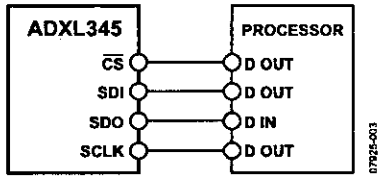


Figure 4. 4-Wire SPI Connection Diagram

To read or write multiple bytes in a single transmission, the multiple-byte bit, located after the R/$\overline{W}$ bit in the first byte transfer (MB in Figure 5 to Figure 7), must be set. After the register addressing and the first byte of data, each subsequent set of clock pulses (eight clock pulses) causes the ADXL345 to point to the next register for a read or write. This shifting continues until the clock pulses cease and $\overline{CS}$ is deasserted. To perform reads or writes on different, nonsequential registers, $\overline{CS}$ must be deasserted between transmissions and the new register must be addressed separately.

The timing diagram for 3-wire SPI reads or writes is shown in Figure 7. The 4-wire equivalents for SPI writes and reads are shown in Figure 5 and Figure 6, respectively.

Table 8. SPI Digital Input/Output Voltage

Parameter	Limit[1]	Unit
Digital Input Voltage		
Low Level Input Voltage (V_{IL})	$0.2 \times V_{DD\ I/O}$	V max
High Level Input Voltage (V_{IH})	$0.8 \times V_{DD\ I/O}$	V min
Digital Output Voltage		
Low Level Output Voltage (V_{OL})	$0.15 \times V_{DD\ I/O}$	V max
High Level Output Voltage (V_{OH})	$0.85 \times V_{DD\ I/O}$	V min

[1] Limits based on characterization results, not production tested.

9. SPI Timing (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)[1]

ıeter	Limit[2, 3] Min	Max	Unit	Description
		5	MHz	SPI clock frequency
	200		ns	1/(SPI clock frequency) mark-space ratio for the SCLK input is 40/60 to 60/40
	10		ns	$\overline{CS}$ falling edge to SCLK falling edge
	10		ns	SCLK rising edge to $\overline{CS}$ rising edge
		100	ns	$\overline{CS}$ rising edge to SDO disabled
	250		ns	$\overline{CS}$ deassertion between SPI communications
	$0.4 \times t_{SCLK}$		ns	SCLK low pulse width (space)
	$0.4 \times t_{SCLK}$		ns	SCLK high pulse width (mark)
		95	ns	SCLK falling edge to SDO transition
	10		ns	SDI valid before SCLK rising edge
	10		ns	SDI valid after SCLK rising edge

, SCLK, SDI, and SDO pins are not internally pulled up or down; they must be driven for proper operation.
based on characterization results, characterized with f_{SCLK} = 5 MHz and bus load capacitance of 100 pF; not production tested.
ning values are measured corresponding to the input thresholds (V_{IL} and V_{IH}) given in Table 8.

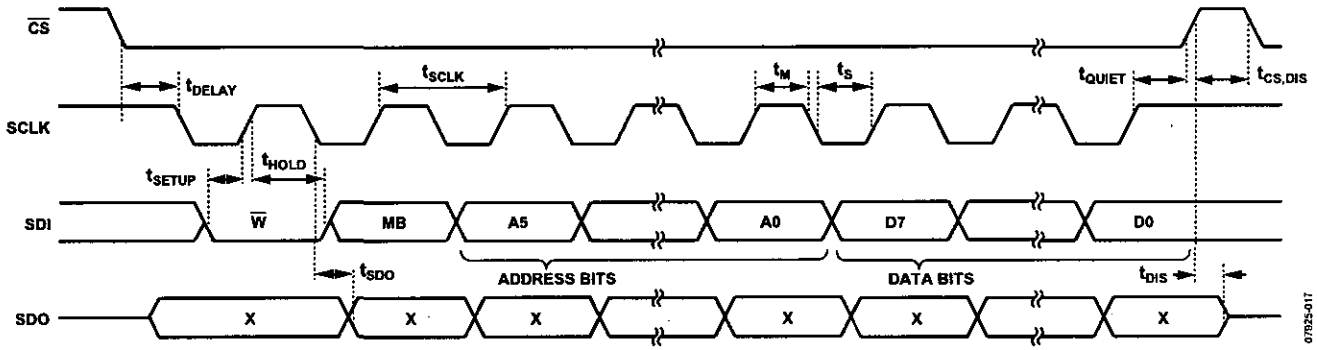


Figure 5. SPI 4-Wire Write

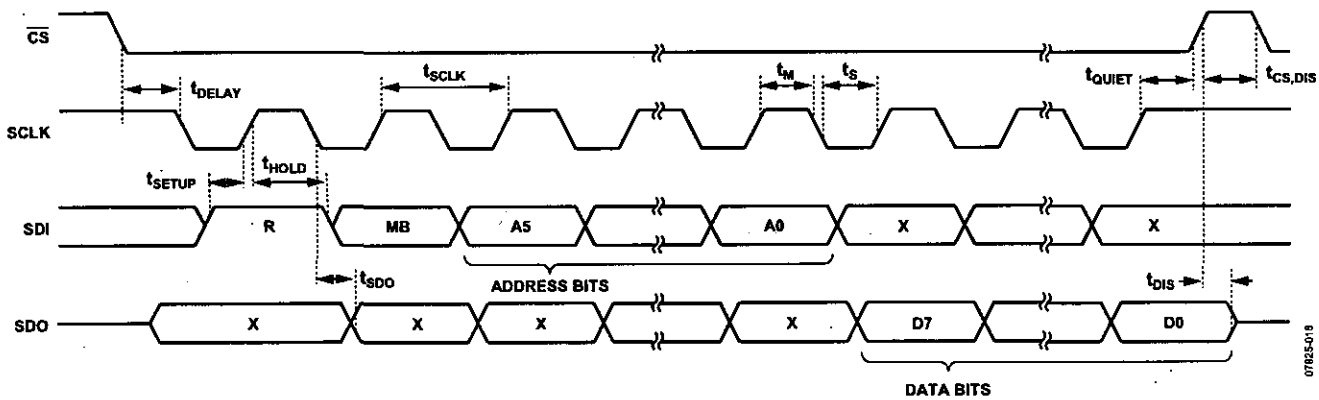
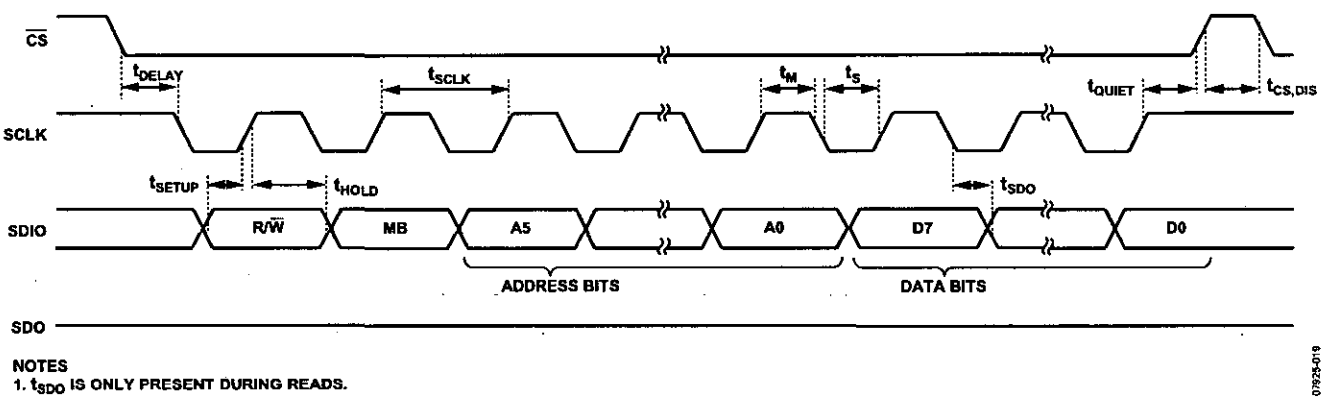


Figure 6. SPI 4-Wire Read



NOTES
1. t_{SDO} IS ONLY PRESENT DURING READS.

Figure 7. SPI 3-Wire Read/Write

$\overline{S}$ tied high to $V_{DD\ I/O}$, the ADXL345 is in I²C mode, ng a simple 2-wire connection as shown in Figure 8. The)45 conforms to the *UM10204 I²C-Bus Specification and anual*, Rev. 03—19 June 2007, available from NXP nductor. It supports standard (100 kHz) and fast (400 kHz) ansfer modes if the timing parameters given in Table 11 ;ure 10 are met. Single- or multiple-byte reads/writes are ted, as shown in Figure 9. With the SDO/ALT ADDRESS h, the 7-bit I²C address for the device is 0x1D, followed by $\overline{W}$ bit. This translates to 0x3A for a write and 0x3B for a read. rnate I²C address of 0x53 (followed by the R/$\overline{W}$ bit) can ;en by grounding the SDO/ALT ADDRESS pin (Pin 12). nslates to 0xA6 for a write and 0xA7 for a read.

If other devices are connected to the same I²C bus, the nominal operating voltage level of these other devices cannot exceed $V_{DD\ I/O}$ by more than 0.3 V. External pull-up resistors, R_P, are necessary for proper I²C operation. Refer to the *UM10204 I²C-Bus Specification and User Manual*, Rev. 03—19 June 2007, when selecting pull-up resistor values to ensure proper operation.

Table 10. I²C Digital Input/Output Voltage

Parameter	Limit[1]	Unit
Digital Input Voltage		
Low Level Input Voltage (V_{IL})	$0.25 \times V_{DD\ I/O}$	V max
High Level Input Voltage (V_{IH})	$0.75 \times V_{DD\ I/O}$	V min
Digital Output Voltage		
Low Level Output Voltage (V_{OL})[2]	$0.2 \times V_{DD\ I/O}$	V max

[1] Limits based on characterization results; not production tested.
[2] The limit given is only for $V_{DD\ I/O} < 2$ V. When $V_{DD\ I/O} > 2$ V, the limit is 0.4 V max.

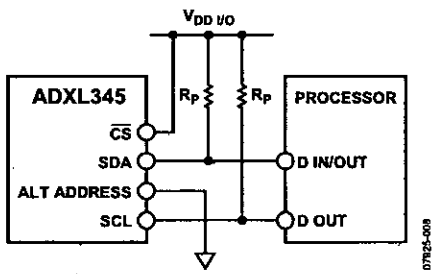


Figure 8. I²C Connection Diagram (Address 0x53)

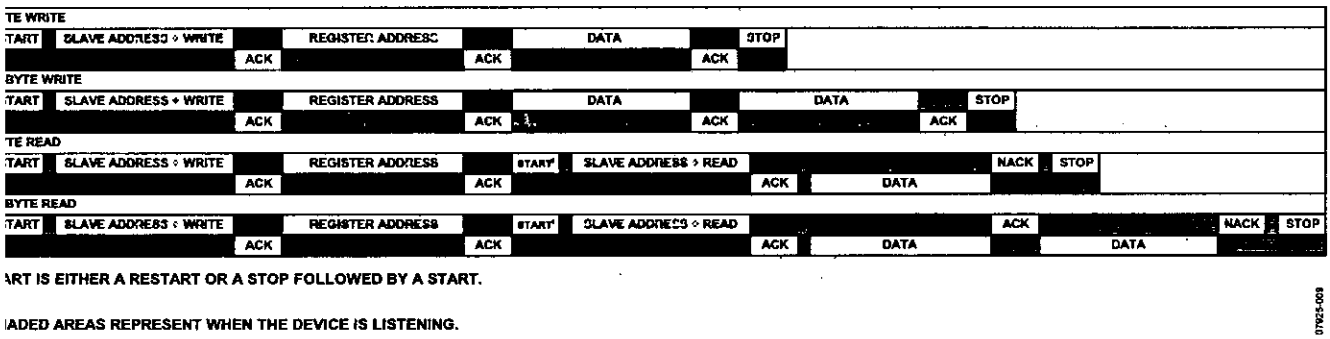


Figure 9. I²C Device Addressing

ble 11. I²C Timing (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

rameter	Limit[1, 2] Min	Max	Unit	Description
		400	kHz	SCL clock frequency
	2.5		µs	SCL cycle time
	0.6		µs	t_{HIGH}, SCL high time
	1.3		µs	t_{LOW}, SCL low time
	0.6		µs	$t_{HD,STA}$, start/repeated start condition hold time
	350		ns	$t_{SU,DAT}$, data setup time
, 4, 5, 6	0	0.65	µs	$t_{HD,DAT}$, data hold time
	0.6		µs	$t_{SU,STA}$, setup time for repeated start
	0.6		µs	$t_{SU,STO}$, stop condition setup time
	1.3		µs	t_{BUF}, bus-free time between a stop condition and a start condition
		300	ns	t_R, rise time of both SCL and SDA when receiving
	0		ns	t_R, rise time of both SCL and SDA when receiving or transmitting
		250	ns	t_F, fall time of SDA when receiving
		300	ns	t_F, fall time of both SCL and SDA when transmitting
	20 + 0.1 C_b[7]		ns	t_F, fall time of both SCL and SDA when transmitting or receiveing
		400	pF	Capacitive load for each bus line

mits based on characterization results, with f_{SCL} = 400 kHz and a 3 mA sink current; not production tested.
i values referred to the V_{IH} and the V_{IL} levels given in Table 10.
is the data hold time that is measured from the falling edge of SCL. It applies to data in transmission and acknowledge times.
transmitting device must internally provide an output hold time of at least 300 ns for the SDA signal (with respect to $V_{IH(min)}$ of the SCL signal) to bridge the undefined region of
e falling edge of SCL.
ıe maximum t_6 value must be met only if the device does not stretch the low period (t_3) of the SCL signal.
ıe maximum value for t_6 is a function of the clock low time (t_3), the clock rise time (t_{10}), and the minimum data setup time ($t_{5(min)}$). This value is calculated as $t_{6(max)} = t_3 - t_{10} - t_{5(min)}$.
, is the total capacitance of one bus line in picofarads.

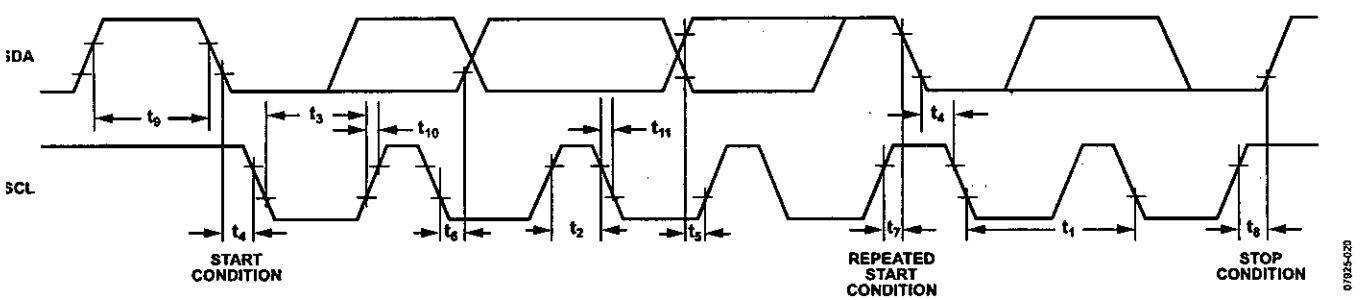


Figure 10. I²C Timing Diagram

IRUPTS

)XL345 provides two output pins for driving interrupts: nd INT2. Each interrupt function is described in detail section. All functions can be used simultaneously, with y limiting feature being that some functions may need e interrupt pins. Interrupts are enabled by setting the riate bit in the INT_ENABLE register (Address 0x2E) mapped to either the INT1 or INT2 pin based on the ts of the INT_MAP register (Address 0x2F). It is recom- d that interrupt bits be configured with the interrupts l, preventing interrupts from being accidentally triggered configuration. This can be done by writing a value of 0x00 NT_ENABLE register. Clearing interrupts is performed y reading the data registers (Address 0x32 to Address 0x37) e interrupt condition is no longer valid for the data-related)ts or by reading the INT_SOURCE register (Address 0x30) emaining interrupts. This section describes the interrupts ı be set in the INT_ENABLE register and monitored in T_SOURCE register.

_READY

ıTA_READY bit is set when new data is available and is when no new data is available.

E_TAP

NGLE_TAP bit is set when a single acceleration event greater than the value in the THRESH_TAP register ss 0x1D) occurs for less time than is specified in R register (Address 0x21).

LE_TAP

)UBLE_TAP bit is set when two acceleration events greater than the value in the THRESH_TAP register ss 0x1D) occur for less time than is specified in the gister (Address 0x21), with the second tap starting after e specified by the latent register (Address 0x22) but within e specified in the window register (Address 0x23). See Detection section for more details.

y

ivity bit is set when acceleration greater than the value n the THRESH_ACT register (Address 0x24) is nced.

ity

ıctivity bit is set when acceleration of less than the value n the THRESH_INACT register (Address 0x25) is experi- or more time than is specified in the TIME_INACT (Address 0x26). The maximum value for TIME_INACT ec.

FALL

EE_FALL bit is set when acceleration of less than the ored in the THRESH_FF register (Address 0x28) is nced for more time than is specified in the TIME_FF (Address 0x29). The FREE_FALL interrupt differs from the inactivity interrupt as follows: all axes always participate, the timer period is much smaller (1.28 sec maximum), and the mode of operation is always dc-coupled.

Watermark

The watermark bit is set when the number of samples in FIFO equals the value stored in the samples bits (Register FIFO_CTL, Address 0x38). The watermark bit is cleared automatically when FIFO is read, and the content returns to a value below the value stored in the samples bits.

Overrun

The overrun bit is set when new data replaces unread data. The precise operation of the overrun function depends on the FIFO mode. In bypass mode, the overrun bit is set when new data replaces unread data in the DATAX, DATAY, and DATAZ registers (Address 0x32 to Address 0x37). In all other modes, the overrun bit is set when FIFO is filled. The overrun bit is automatically cleared when the contents of FIFO are read.

FIFO

The ADXL345 contains patent pending technology for an embedded 32-level FIFO that can be used to minimize host processor burden. This buffer has four modes: bypass, FIFO, stream, and trigger (see Table 19). Each mode is selected by the settings of the FIFO_MODE bits in the FIFO_CTL register (Address 0x38).

Bypass Mode

In bypass mode, FIFO is not operational and, therefore, remains empty.

FIFO Mode

In FIFO mode, data from measurements of the x-, y-, and z-axes are stored in FIFO. When the number of samples in FIFO equals the level specified in the samples bits of the FIFO_CTL register (Address 0x38), the watermark interrupt is set. FIFO continues accumulating samples until it is full (32 samples from measurements of the x-, y-, and z-axes) and then stops collecting data. After FIFO stops collecting data, the device continues to operate; therefore, features such as tap detection can be used after FIFO is full. The watermark interrupt continues to occur until the number of samples in FIFO is less than the value stored in the samples bits of the FIFO_CTL register.

Stream Mode

In stream mode, data from measurements of the x-, y-, and z-axes are stored in FIFO. When the number of samples in FIFO equals the level specified in the samples bits of the FIFO_CTL register (Address 0x38), the watermark interrupt is set. FIFO continues accumulating samples and holds the latest 32 samples from measurements of the x-, y-, and z-axes, discarding older data as new data arrives. The watermark interrupt continues occurring until the number of samples in FIFO is less than the value stored in the samples bits of the FIFO_CTL register.

'rigger Mode

ı trigger mode, FIFO accumulates samples, holding the latest 2 samples from measurements of the x-, y-, and z-axes. After trigger event occurs and an interrupt is sent to the INT1 or NT2 pin (determined by the trigger bit in the FIFO_CTL register), IFO keeps the last n samples (where n is the value specified by ıe samples bits in the FIFO_CTL register) and then operates in IFO mode, collecting new samples only when FIFO is not full. ı delay of at least 5 µs should be present between the trigger event ccurring and the start of reading data from the FIFO to allow ıe FIFO to discard and retain the necessary samples. Additional igger events cannot be recognized until the trigger mode is set. To reset the trigger mode, set the device to bypass mode ıd then set the device back to trigger mode. Note that the IFO data should be read first because placing the device into ypass mode clears FIFO.

etrieving Data from FIFO

he FIFO data is read through the DATAX, DATAY, and DATAZ gisters (Address 0x32 to Address 0x37). When the FIFO is in IFO, stream, or trigger mode, reads to the DATAX, DATAY, ıd DATAZ registers read data stored in the FIFO. Each time ata is read from the FIFO, the oldest x-, y-, and z-axes data are laced into the DATAX, DATAY and DATAZ registers.

a single-byte read operation is performed, the remaining ytes of data for the current FIFO sample are lost. Therefore, all ces of interest should be read in a burst (or multiple-byte) read peration. To ensure that the FIFO has completely popped (that , that new data has completely moved into the DATAX, DATAY, ıd DATAZ registers), there must be at least 5 µs between the ıd of reading the data registers and the start of a new read of ıe FIFO or a read of the FIFO_STATUS register (Address 0x39). he end of reading a data register is signified by the transition om Register 0x37 to Register 0x38 or by the $\overline{CS}$ pin going high.

or SPI operation at 1.6 MHz or less, the register addressing ortion of the transmission is a sufficient delay to ensure that ıe FIFO has completely popped. For SPI operation greater than 6 MHz, it is necessary to deassert the $\overline{CS}$ pin to ensure a total elay of 5 µs; otherwise, the delay will not be sufficient. The total elay necessary for 5 MHz operation is at most 3.4 µs. This is ot a concern when using I²C mode because the communication ıte is low enough to ensure a sufficient delay between FIFO reads.

SELF-TEST

The ADXL345 incorporates a self-test feature that effectively tests its mechanical and electronic systems simultaneously. When the self-test function is enabled (via the SELF_TEST bit in the DATA_FORMAT register, Address 0x31), an electrostati force is exerted on the mechanical sensor. This electrostatic force moves the mechanical sensing element in the same manner as acceleration, and it is additive to the acceleration experienced by the device. This added electrostatic force results in an outpu change in the x-, y-, and z-axes. Because the electrostatic force is proportional to V_S^2, the output change varies with V_S. The self-test feature of the ADXL345 also exhibits a bimodal behavior that depends on which phase of the clock self-test is enabled. However, the limits shown in Table 1 and Table 12 to Table 15 are valid for all potential self-test values across the entire allowable voltage range. Use of the self-test feature at data rates less than 100 Hz may yield values outside these limits. Therefore, the part should be placed into a data rate of 100 Hz or greater when using self-test.

Table 12. Self-Test Output in LSB for ±2 *g*, Full Resolution

Axis	Min	Max	Unit
X	50	540	LSB
Y	−540	−50	LSB
Z	75	875	LSB

Table 13. Self-Test Output in LSB for ±4 *g*, 10-Bit Resolutio

Axis	Min	Max	Unit
X	25	270	LSB
Y	−270	−25	LSB
Z	38	438	LSB

Table 14. Self-Test Output in LSB for ±8 *g*, 10-Bit Resolutio

Axis	Min	Max	Unit
X	12	135	LSB
Y	−135	−12	LSB
Z	19	219	LSB

Table 15. Self-Test Output in LSB for ±16 *g*, 10-Bit Resolutio

Axis	Min	Max	Unit
X	6	67	LSB
Y	−67	−6	LSB
Z	10	110	LSB

ISTER MAP

16. Register Map

Address					
	Dec	Name	Type	Reset Value	Description
	0	DEVID	R	11100101	Device ID.
ɔ 0x01C	1 to 28	Reserved			Reserved. Do not access.
	29	THRESH_TAP	R/$\overline{W}$	00000000	Tap threshold.
	30	OFSX	R/$\overline{W}$	00000000	X-axis offset.
	31	OFSY	R/$\overline{W}$	00000000	Y-axis offset.
	32	OFSZ	R/$\overline{W}$	00000000	Z-axis offset.
	33	DUR	R/$\overline{W}$	00000000	Tap duration.
	34	Latent	R/$\overline{W}$	00000000	Tap latency.
	35	Window	R/$\overline{W}$	00000000	Tap window.
	36	THRESH_ACT	R/$\overline{W}$	00000000	Activity threshold.
	37	THRESH_INACT	R/$\overline{W}$	00000000	Inactivity threshold.
	38	TIME_INACT	R/$\overline{W}$	00000000	Inactivity time.
	39	ACT_INACT_CTL	R/$\overline{W}$	00000000	Axis enable control for activity and inactivity detection.
	40	THRESH_FF	R/$\overline{W}$	00000000	Free-fall threshold.
	41	TIME_FF	R/$\overline{W}$	00000000	Free-fall time.
	42	TAP_AXES	R/$\overline{W}$	00000000	Axis control for tap/double tap.
	43	ACT_TAP_STATUS	R	00000000	Source of tap/double tap.
	44	BW_RATE	R/$\overline{W}$	00001010	Data rate and power mode control.
	45	POWER_CTL	R/$\overline{W}$	00000000	Power-saving features control.
	46	INT_ENABLE	R/$\overline{W}$	00000000	Interrupt enable control.
	47	INT_MAP	R/$\overline{W}$	00000000	Interrupt mapping control.
	48	INT_SOURCE	R	00000010	Source of interrupts.
	49	DATA_FORMAT	R/$\overline{W}$	00000000	Data format control.
	50	DATAX0	R	00000000	X-Axis Data 0.
	51	DATAX1	R	00000000	X-Axis Data 1.
	52	DATAY0	R	00000000	Y-Axis Data 0.
	53	DATAY1	R	00000000	Y-Axis Data 1.
	54	DATAZ0	R	00000000	Z-Axis Data 0.
	55	DATAZ1	R	00000000	Z-Axis Data 1.
	56	FIFO_CTL	R/$\overline{W}$	00000000	FIFO control.
	57	FIFO_STATUS	R	00000000	FIFO status.

EGISTER DEFINITIONS

gister 0x00—DEVID (Read Only)

	D6	D5	D4	D3	D2	D1	D0
	1	1	0	0	1	0	1

ıe DEVID register holds a fixed device ID code of 0xE5 45 octal).

gister 0x1D—THRESH_TAP (Read/Write)

ıe THRESH_TAP register is eight bits and holds the threshold lue for tap interrupts. The data format is unsigned, so the ıgnitude of the tap event is compared with the value in HRESH_TAP. The scale factor is 62.5 mg/LSB (that is, 0xFF = 6 *g*). A value of 0 may result in undesirable behavior if tap/ uble tap interrupts are enabled.

gister 0x1E, Register 0x1F, Register 0x20—OFSX, SY, OFSZ (Read/Write)

ıe OFSX, OFSY, and OFSZ registers are each eight bits and er user-set offset adjustments in twos complement format th a scale factor of 15.6 mg/LSB (that is, 0x7F = +2 *g*).

gister 0x21—DUR (Read/Write)

ıe DUR register is eight bits and contains an unsigned time lue representing the maximum time that an event must be ove the THRESH_TAP threshold to qualify as a tap event. The ıle factor is 625 µs/LSB. A value of 0 disables the tap/double › functions.

gister 0x22—Latent (Read/Write)

ıe latent register is eight bits and contains an unsigned time lue representing the wait time from the detection of a tap event the start of the time window (defined by the window register) ring which a possible second tap event can be detected. The scale tor is 1.25 ms/LSB. A value of 0 disables the double tap function.

gister 0x23—Window (Read/Write)

ıe window register is eight bits and contains an unsigned time lue representing the amount of time after the expiration of the ency time (determined by the latent register) during which a cond valid tap can begin. The scale factor is 1.25 ms/LSB. A lue of 0 disables the double tap function.

gister 0x24—THRESH_ACT (Read/Write)

ıe THRESH_ACT register is eight bits and holds the threshold lue for detecting activity. The data format is unsigned, so the ıgnitude of the activity event is compared with the value in e THRESH_ACT register. The scale factor is 62.5 mg/LSB. value of 0 may result in undesirable behavior if the activity errupt is enabled.

gister 0x25—THRESH_INACT (Read/Write)

ıe THRESH_INACT register is eight bits and holds the threshold lue for detecting inactivity. The data format is unsigned, so e magnitude of the inactivity event is compared with the value the THRESH_INACT register. The scale factor is 62.5 mg/LSB. value of 0 mg may result in undesirable behavior if the inactivity errupt is enabled.

Register 0x26—TIME_INACT (Read/Write)

The TIME_INACT register is eight bits and contains an unsigned time value representing the amount of time that acceleration must be less than the value in the THRESH_INACT register for inactivity to be declared. The scale factor is 1 sec/LSB. Unlike the other interrupt functions, which use unfiltered data (see the Threshold section), the inactivity function uses filtered output data. At least one output sample must be generated for the inactivity interrupt to be triggered. This results in the function appearing unresponsive if the TIME_INACT register is set to a value less than the time constant of the output data rate. A valu of 0 results in an interrupt when the output data is less than the value in the THRESH_INACT register.

Register 0x27—ACT_INACT_CTL (Read/Write)

D7	D6	D5	D4
ACT ac/dc	ACT_X enable	ACT_Y enable	ACT_Z enable
D3	**D2**	**D1**	**D0**
INACT ac/dc	INACT_X enable	INACT_Y enable	INACT_Z enabl

ACT AC/DC and INACT AC/DC Bits

A setting of 0 selects dc-coupled operation, and a setting of 1 enables ac-coupled operation. In dc-coupled operation, the current acceleration magnitude is compared directly with THRESH_ACT and THRESH_INACT to determine whether activity or inactivity is detected.

In ac-coupled operation for activity detection, the acceleration value at the start of activity detection is taken as a reference value. New samples of acceleration are then compared to this reference value, and if the magnitude of the difference exceeds the THRESH_ACT value, the device triggers an activity interrupt.

Similarly, in ac-coupled operation for inactivity detection, a reference value is used for comparison and is updated whenev the device exceeds the inactivity threshold. After the reference value is selected, the device compares the magnitude of the difference between the reference value and the current acceleratio with THRESH_INACT. If the difference is less than the value i THRESH_INACT for the time in TIME_INACT, the device is considered inactive and the inactivity interrupt is triggered.

ACT_x Enable Bits and INACT_x Enable Bits

A setting of 1 enables x-, y-, or z-axis participation in detecting activity or inactivity. A setting of 0 excludes the selected axis from participation. If all axes are excluded, the function is disabled.

Register 0x28—THRESH_FF (Read/Write)

The THRESH_FF register is eight bits and holds the threshold value, in unsigned format, for free-fall detection. The root-sum-square (RSS) value of all axes is calculated and compared with the value in THRESH_FF to determine if a free-fall event occurre The scale factor is 62.5 mg/LSB. Note that a value of 0 mg ma result in undesirable behavior if the free-fall interrupt is enabled. Values between 300 mg and 600 mg (0x05 to 0x09) are recommended.

er 0x29—TIME_FF (Read/Write)

ME_FF register is eight bits and stores an unsigned time presenting the minimum time that the RSS value of all axes e less than THRESH_FF to generate a free-fall interrupt. le factor is 5 ms/LSB. A value of 0 may result in undesirable or if the free-fall interrupt is enabled. Values between 100 ms 0 ms (0x14 to 0x46) are recommended.

er 0x2A—TAP_AXES (Read/Write)

D6	D5	D4	D3	D2	D1	D0
)	0	0	Suppress	TAP_X enable	TAP_Y enable	TAP_Z enable

ss Bit

the suppress bit suppresses double tap detection if tion greater than the value in THRESH_TAP is present n taps. See the Tap Detection section for more details.

Enable Bits

ig of 1 in the TAP_X enable, TAP_Y enable, or TAP_Z bit enables x-, y-, or z-axis participation in tap detection. ig of 0 excludes the selected axis from participation in ection.

er 0x2B—ACT_TAP_STATUS (Read Only)

6	D5	D4	D3	D2	D1	D0
T_X urce	ACT_Y source	ACT_Z source	Asleep	TAP_X source	TAP_Y source	TAP_Z source

Source and TAP_x Source Bits

its indicate the first axis involved in a tap or activity A setting of 1 corresponds to involvement in the event, etting of 0 corresponds to no involvement. When new available, these bits are not cleared but are overwritten by v data. The ACT_TAP_STATUS register should be read learing the interrupt. Disabling an axis from participation he corresponding source bit when the next activity or ible tap event occurs.

Bit

ig of 1 in the asleep bit indicates that the part is asleep, etting of 0 indicates that the part is not asleep. See the r 0x2D—POWER_CTL (Read/Write) section for more ation on autosleep mode.

er 0x2C—BW_RATE (Read/Write)

D6	D5	D4	D3	D2	D1	D0
)	0	LOW_POWER	Rate			

POWER Bit

g of 0 in the LOW_POWER bit selects normal operation, etting of 1 selects reduced power operation, which has at higher noise (see the Power Modes section for details).

Rate Bits

These bits select the device bandwidth and output data rate (see Table 6 and Table 7 for details). The default value is 0x0A, which translates to a 100 Hz output data rate. An output data rate should be selected that is appropriate for the communication protocol and frequency selected. Selecting too high of an output data rate with a low communication speed results in samples being discarded.

Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

Link Bit

A setting of 1 in the link bit with both the activity and inactivity functions enabled delays the start of the activity function until inactivity is detected. After activity is detected, inactivity detection begins, preventing the detection of activity. This bit serially links the activity and inactivity functions. When this bit is set to 0, the inactivity and activity functions are concurrent. Additional information can be found in the Link Mode section.

When clearing the link bit, it is recommended that the part be placed into standby mode and then set back to measurement mode with a subsequent write. This is done to ensure that the device is properly biased if sleep mode is manually disabled; otherwise, the first few samples of data after the link bit is cleared may have additional noise, especially if the device was asleep when the bit was cleared.

AUTO_SLEEP Bit

If the link bit is set, a setting of 1 in the AUTO_SLEEP bit sets the ADXL345 to switch to sleep mode when inactivity is detected (that is, when acceleration has been below the THRESH_INACT value for at least the time indicated by TIME_INACT). A setting of 0 disables automatic switching to sleep mode. See the description of the sleep bit in this section for more information.

When clearing the AUTO_SLEEP bit, it is recommended that the part be placed into standby mode and then set back to measurement mode with a subsequent write. This is done to ensure that the device is properly biased if sleep mode is manually disabled; otherwise, the first few samples of data after the AUTO_SLEEP bit is cleared may have additional noise, especially if the device was asleep when the bit was cleared.

Measure Bit

A setting of 0 in the measure bit places the part into standby mode, and a setting of 1 places the part into measurement mode. The ADXL345 powers up in standby mode with minimum power consumption.

ɛep Bit

ɜetting of 0 in the sleep bit puts the part into the normal mode operation, and a setting of 1 places the part into sleep mode. ɛp mode suppresses DATA_READY, stops transmission of data FIFO, and switches the sampling rate to one specified by the keup bits. In sleep mode, only the activity function can be used.

hen clearing the sleep bit, it is recommended that the part be ıced into standby mode and then set back to measurement ɔde with a subsequent write. This is done to ensure that the vice is properly biased if sleep mode is manually disabled; ıerwise, the first few samples of data after the sleep bit is ıared may have additional noise, especially if the device was eep when the bit was cleared.

ıkeup Bits

ese bits control the frequency of readings in sleep mode as ɜcribed in Table 17.

ble 17. Frequency of Readings in Sleep Mode

Setting		
	D0	Frequency (Hz)
	0	8
	1	4
	0	2
	1	1

gister 0x2E—INT_ENABLE (Read/Write)

	D6	D5	D4
ıTA_READY	SINGLE_TAP	DOUBLE_TAP	Activity
	D2	D1	D0
ıctivity	FREE_FALL	Watermark	Overrun

ɪting bits in this register to a value of 1 enables their respective ıctions to generate interrupts, whereas a value of 0 prevents ɜ functions from generating interrupts. The DATA_READY, ıtermark, and overrun bits enable only the interrupt output; ɜ functions are always enabled. It is recommended that interrupts configured before enabling their outputs.

gister 0x2F—INT_MAP (Read/Write)

	D6	D5	D4
ıTA_READY	SINGLE_TAP	DOUBLE_TAP	Activity
	D2	D1	D0
ıctivity	FREE_FALL	Watermark	Overrun

ıy bits set to 0 in this register send their respective interrupts to ɜ INT1 pin, whereas bits set to 1 send their respective interrupts the INT2 pin. All selected interrupts for a given pin are ORed.

gister 0x30—INT_SOURCE (Read Only)

	D6	D5	D4
ıTA_READY	SINGLE_TAP	DOUBLE_TAP	Activity
	D2	D1	D0
ıctivity	FREE_FALL	Watermark	Overrun

Bits set to 1 in this register indicate that their respective functions have triggered an event, whereas a value of 0 indicates that the corresponding event has not occurred. The DATA_READY, watermark, and overrun bits are always set if the corresponding events occur, regardless of the INT_ENABLE register settings, and are cleared by reading data from the DATAX, DATAY, and DATAZ registers. The DATA_READY and watermark bits may require multiple reads, as indicated in the FIFO mode descriptions in the FIFO section. Other bits, and the corresponding interrupts, are cleared by reading the INT_SOURCE register.

Register 0x31—DATA_FORMAT (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

The DATA_FORMAT register controls the presentation of data to Register 0x32 through Register 0x37. All data, except that for the ±16 *g* range, must be clipped to avoid rollover.

SELF_TEST Bit

A setting of 1 in the SELF_TEST bit applies a self-test force to the sensor, causing a shift in the output data. A value of 0 disables the self-test force.

SPI Bit

A value of 1 in the SPI bit sets the device to 3-wire SPI mode, and a value of 0 sets the device to 4-wire SPI mode.

INT_INVERT Bit

A value of 0 in the INT_INVERT bit sets the interrupts to activ high, and a value of 1 sets the interrupts to active low.

FULL_RES Bit

When this bit is set to a value of 1, the device is in full resolution mode, where the output resolution increases with the *g* range set by the range bits to maintain a 4 mg/LSB scale factor. When the FULL_RES bit is set to 0, the device is in 10-bit mode, and the range bits determine the maximum *g* range and scale factor

Justify Bit

A setting of 1 in the justify bit selects left (MSB) justified mode and a setting of 0 selects right justified mode with sign extension.

Range Bits

These bits set the *g* range as described in Table 18.

Table 18. *g* Range Setting

Setting		
D1	D0	*g* Range
0	0	±2 *g*
0	1	±4 *g*
1	0	±8 *g*
1	1	±16 *g*

ter 0x32 to Register 0x37—DATAX0, DATAX1, Y0, DATAY1, DATAZ0, DATAZ1 (Read Only)

six bytes (Register 0x32 to Register 0x37) are eight bits nd hold the output data for each axis. Register 0x32 and :r 0x33 hold the output data for the x-axis, Register 0x34 and :r 0x35 hold the output data for the y-axis, and Register 0x36 :gister 0x37 hold the output data for the z-axis. The output twos complement, with DATAx0 as the least significant ıd DATAx1 as the most significant byte, where x represent X, :. The DATA_FORMAT register (Address 0x31) controls mat of the data. It is recommended that a multiple-byte f all registers be performed to prevent a change in data :n reads of sequential registers.

ter 0x38—FIFO_CTL (Read/Write)

	D6	D5	D4	D3	D2	D1	D0
ıODE	Trigger	Samples					

_MODE Bits

bits set the FIFO mode, as described in Table 19.

19. FIFO Modes

ıg)6	Mode	Function
)	Bypass	FIFO is bypassed.
	FIFO	FIFO collects up to 32 values and then stops collecting data, collecting new data only when FIFO is not full.
)	Stream	FIFO holds the last 32 data values. When FIFO is full, the oldest data is overwritten with newer data.
	Trigger	When triggered by the trigger bit, FIFO holds the last data samples before the trigger event and then continues to collect data until full. New data is collected only when FIFO is not full.

:r Bit

: of 0 in the trigger bit links the trigger event of trigger mode '1, and a value of 1 links the trigger event to INT2.

Samples Bits

The function of these bits depends on the FIFO mode selected (see Table 20). Entering a value of 0 in the samples bits immediately sets the watermark status bit in the INT_SOURCE register, regardless of which FIFO mode is selected. Undesirable operation may occur if a value of 0 is used for the samples bits when trigger mode is used.

Table 20. Samples Bits Functions

FIFO Mode	Samples Bits Function
Bypass	None.
FIFO	Specifies how many FIFO entries are needed to trigger a watermark interrupt.
Stream	Specifies how many FIFO entries are needed to trigger a watermark interrupt.
Trigger	Specifies how many FIFO samples are retained in the FIFO buffer before a trigger event.

0x39—FIFO_STATUS (Read Only)

D7	D6	D5	D4	D3	D2	D1	D0
FIFO_TRIG	0	Entries					

FIFO_TRIG Bit

A 1 in the FIFO_TRIG bit corresponds to a trigger event occurring, and a 0 means that a FIFO trigger event has not occurred.

Entries Bits

These bits report how many data values are stored in FIFO. Access to collect the data from FIFO is provided through the DATAX, DATAY, and DATAZ registers. FIFO reads must be done in burst or multiple-byte mode because each FIFO level is cleared after any read (single- or multiple-byte) of FIFO. FIFO stores a maximum of 32 entries, which equates to a maximum of 33 entries available at any given time because an additional entry is available at the output filter of the device.

PPLICATIONS INFORMATION

OWER SUPPLY DECOUPLING

1 µF tantalum capacitor (C_S) at V_S and a 0.1 µF ceramic capacitor $_{IO}$) at $V_{DD\ I/O}$ placed close to the ADXL345 supply pins is used r testing and is recommended to adequately decouple the celerometer from noise on the power supply. If additional coupling is necessary, a resistor or ferrite bead, no larger than 0 Ω, in series with V_S may be helpful. Additionally, increasing e bypass capacitance on V_S to a 10 µF tantalum capacitor in rallel with a 0.1 µF ceramic capacitor may also improve noise.

re should be taken to ensure that the connection from the XL345 ground to the power supply ground has low impedance cause noise transmitted through ground has an effect similar noise transmitted through V_S. It is recommended that V_S and $_{DD\ I/O}$ be separate supplies to minimize digital clocking noise the V_S supply. If this is not possible, additional filtering of e supplies as previously mentioned may be necessary.

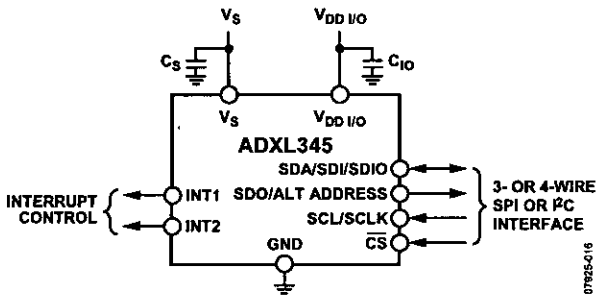


Figure 11. Application Diagram

ECHANICAL CONSIDERATIONS FOR MOUNTING

e ADXL345 should be mounted on the PCB in a location se to a hard mounting point of the PCB to the case. Mounting e ADXL345 at an unsupported PCB location, as shown in gure 12, may result in large, apparent measurement errors due undampened PCB vibration. Locating the accelerometer near ard mounting point ensures that any PCB vibration at the celerometer is above the accelerometer's mechanical sensor sonant frequency and, therefore, effectively invisible to the celerometer.

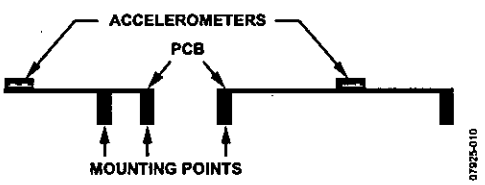


Figure 12. Incorrectly Placed Accelerometers

AP DETECTION

e tap interrupt function is capable of detecting either single double taps. The following parameters are shown in Figure 13 : a valid single and valid double tap event:

- The tap detection threshold is defined by the THRESH_TAP register (Address 0x1D).
- The maximum tap duration time is defined by the DUR register (Address 0x21).
- The tap latency time is defined by the latent register (Address 0x22) and is the waiting period from the end of the first tap until the start of the time window, when a second tap can be detected, which is determined by the value in the window register (Address 0x23).
- The interval after the latency time (set by the latent register) i defined by the window register. Although a second tap must begin after the latency time has expired, it need not finish before the end of the time defined by the window register.

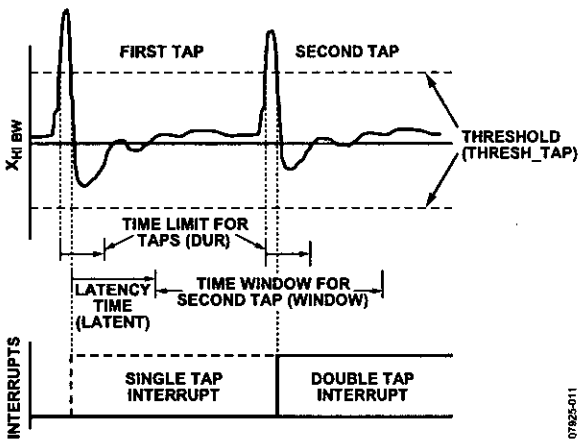


Figure 13. Tap Interrupt Function with Valid Single and Double Taps

If only the single tap function is in use, the single tap interrupt is triggered when the acceleration goes below the threshold, as long as DUR has not been exceeded. If both single and double tap functions are in use, the single tap interrupt is triggered when the double tap event has been either validated or invalidated.

Several events can occur to invalidate the second tap of a double tap event. First, if the suppress bit in the TAP_AXES register (Address 0x2A) is set, any acceleration spike above the threshold during the latency time (set by the latent register) invalidates the double tap detection, as shown in Figure 14.

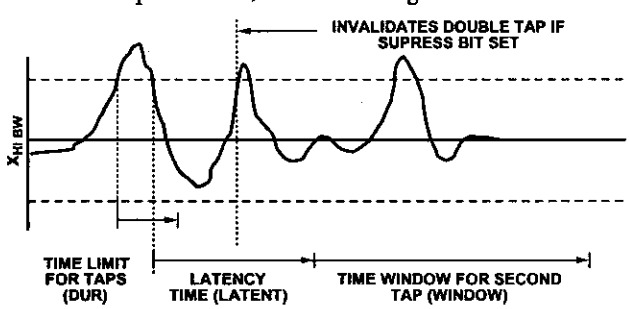


Figure 14. Double Tap Event Invalid Due to High g Event When the Suppress Bit Is Set

A double tap event can also be invalidated if acceleration above the threshold is detected at the start of the time window for the second tap (set by the window register). This results in an invalid double tap at the start of this window, as shown in Figure 15. Additionally, a double tap event can be invalidated if an accel-

exceeds the time limit for taps (set by the DUR register), ıg in an invalid double tap at the end of the DUR time r the second tap event, also shown in Figure 15.

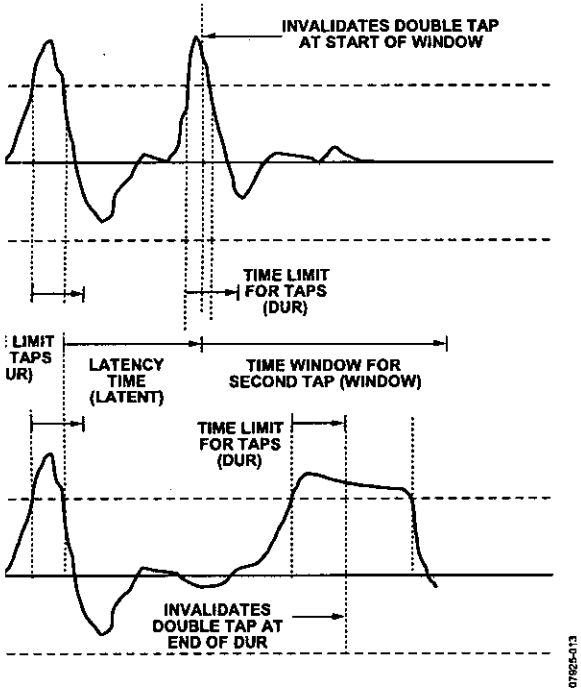


Figure 15. Tap Interrupt Function with Invalid Double Taps

aps, double taps, or both can be detected by setting the ive bits in the INT_ENABLE register (Address 0x2E). over participation of each of the three axes in single tap/ tap detection is exerted by setting the appropriate bits in ?_AXES register (Address 0x2A). For the double tap ı to operate, both the latent and window registers must ɔ a nonzero value.

ıechanical system has somewhat different single tap/double ɔonses based on the mechanical characteristics of the Therefore, some experimentation with values for the vindow, and THRESH_TAP registers is required. In , a good starting point is to set the latent register to a reater than 0x10, to set the window register to a value than 0x10, and to set the THRESH_TAP register to be than 3 *g*. Setting a very low value in the latent, window, or H_TAP register may result in an unpredictable response he accelerometer picking up echoes of the tap inputs.

tap interrupt has been received, the first axis to exceed RESH_TAP level is reported in the ACT_TAP_STATUS (Address 0x2B). This register is never cleared, but is itten with new data.

SHOLD

ver output data rates are achieved by decimating a ın sampling frequency inside the device. The activity, l, and single tap/double tap detection functions are ned using unfiltered data. Since the output data is , the high frequency and high *g* data that is used to determine activity, free-fall, and single tap/double tap events may not be present if the output of the accelerometer is examined. This may result in trigger events being detected when acceleration does not appear to trigger an event because the unfiltered data may have exceeded a threshold or remained below a threshold for a certain period of time while the filtered output data has not exceeded such a threshold.

LINK MODE

The function of the link bit is to reduce the number of activity interrupts that the processor must service by setting the device to look for activity only after inactivity. For proper operation of this feature, the processor must still respond to the activity and inactivity interrupts by reading the INT_SOURCE register (Address 0x30) and, therefore, clearing the interrupts. If an activity interrupt is not cleared, the part cannot go into autosleep mode. The asleep bit in the ACT_TAP_STATUS register (Address 0x2B) indicates if the part is asleep.

SLEEP MODE VS. LOW POWER MODE

In applications where a low data rate is sufficient and low power consumption is desired, it is recommended that the low power mode be used in conjunction with the FIFO. The sleep mode, while offering a low data rate and low average current consumption, suppresses the DATA_READY interrupt, preventing the accelerometer from sending an interrupt signal to the host processor when data is ready to be collected. In this application, setting the part into low power mode (by setting the LOW_POWER bit in the BW_RATE register) and enabling the FIFO in FIFO mode to collect a large value of samples reduces the power consumption of the ADXL345 and allows the host processor to go to sleep while the FIFO is filling up.

USING SELF-TEST

The self-test change is defined as the difference between the acceleration output of an axis with self-test enabled and the acceleration output of the same axis with self-test disabled (see Endnote 4 of Table 1). This definition assumes that the sensor does not move between these two measurements, because if the sensor moves, a non–self-test related shift corrupts the test.

Proper configuration of the ADXL345 is also necessary for an accurate self-test measurement. The part should be set with a data rate greater than or equal to 100 Hz. This is done by ensuring that a value greater than or equal to 0x0A is written into the rate bits (Bit D3 through Bit D0) in the BW_RATE register (Address 0x2C). It is also recommended that the part be set to full-resolution, 16 *g* mode to ensure that there is sufficient dynamic range for the entire self-test shift. This is done by setting Bit D3 of the DATA_FORMAT register (Address 0x31) and writing a value of 0x03 to the range bits (Bit D1 and Bit D0) of the DATA_FORMAT register (Address 0x31). This results in a high dynamic range for measurement and a 3.9 mg/LSB scale factor.

After the part is configured for accurate self-test measurement, several samples of x-, y-, and z-axis acceleration data should be retrieved from the sensor and averaged together. The number of

nples averaged is a choice of the system designer, but a recom-
ended starting point is 0.1 sec worth of data, which corresponds
10 samples at 100 Hz data rate. The averaged values should
stored and labeled appropriately as the self-test disabled data,
it is, X_{ST_OFF}, Y_{ST_OFF}, and Z_{ST_OFF}.

xt, self-test should be enabled by setting Bit D7 of the
ATA_FORMAT register (Address 0x31). The output needs
ne time (about four samples) to settle after enabling self-test.
ter allowing the output to settle, several samples of the x-, y-,
d z-axis acceleration data should be taken again and averaged. It
ecommended that the same number of samples be taken for
s average as was previously taken. These averaged values should
in be stored and labeled appropriately as the value with self-
t enabled, that is, X_{ST_ON}, Y_{ST_ON}, and Z_{ST_ON}. Self-test can then
disabled by clearing Bit D7 of the DATA_FORMAT register
ddress 0x31).

ith the stored values for self-test enabled and disabled, the
f-test change is as follows:

$$X_{ST} = X_{ST\_ON} - X_{ST\_OFF}$$

$$Y_{ST} = Y_{ST\_ON} - Y_{ST\_OFF}$$

$$Z_{ST} = Z_{ST\_ON} - Z_{ST\_OFF}$$

Because the measured output for each axis is expressed in LSBs, X_{ST}, Y_{ST}, and Z_{ST} are also expressed in LSBs. These values can be converted to *g*'s of acceleration by multiplying each value by the 3.9 mg/LSB scale factor, if configured for full-resolution, 16 *g* mode. Additionally, Table 12 through Table 15 correspond to the self-test range converted to LSBs and can be compared with the measured self-test change. If the part was placed into full-resolution, 16 *g* mode, the values listed in Table 12 should be used. Although the fixed 10-bit mode or a range other than 16 *g* can be used, a different set of values, as indicated in Table 13 through Table 15, would need to be used. Using a range below 8 *g* may result in insufficient dynamic range and should be considered when selecting the range of operation for measuring self-test. In addition, note that the range in Table 1 and the values in Table 12 through Table 15 take into account all possible supply voltages, V_S, and no additional conversion due to V_S is necessary.

If the self-test change is within the valid range, the test is considered successful. Generally, a part is considered to pass if the minimum magnitude of change is achieved. However, a part that changes by more than the maximum magnitude is not necessarily a failure.

OF ACCELERATION SENSITIVITY

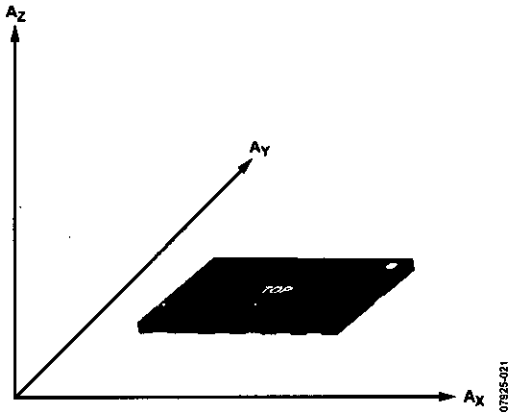


Figure 16. Axes of Acceleration Sensitivity (Corresponding Output Voltage Increases When Accelerated Along the Sensitive Axis)

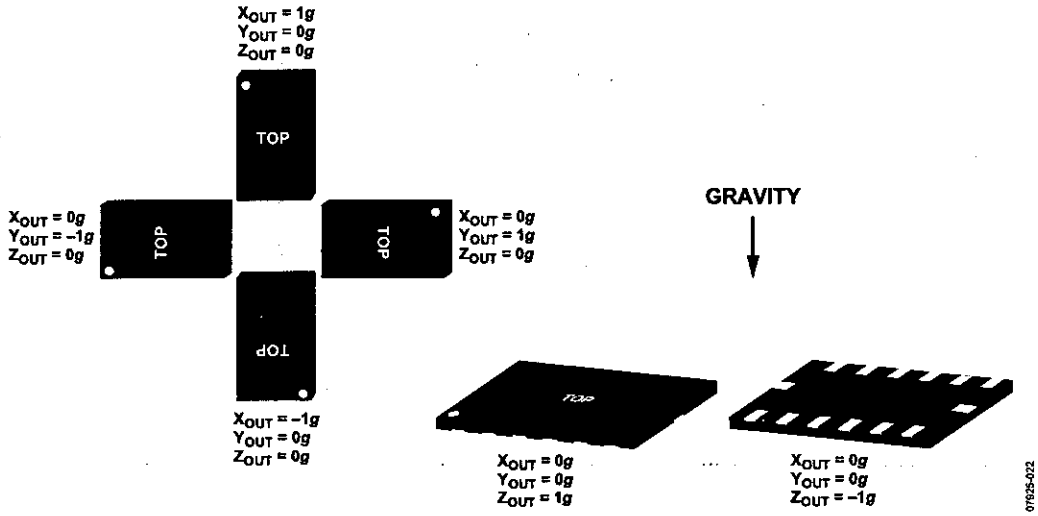


Figure 17. Output Response vs. Orientation to Gravity

LAYOUT AND DESIGN RECOMMENDATIONS

Figure 18 shows the recommended printed wiring board land pattern. Figure 19 and Table 21 provide details about the recommen soldering profile.

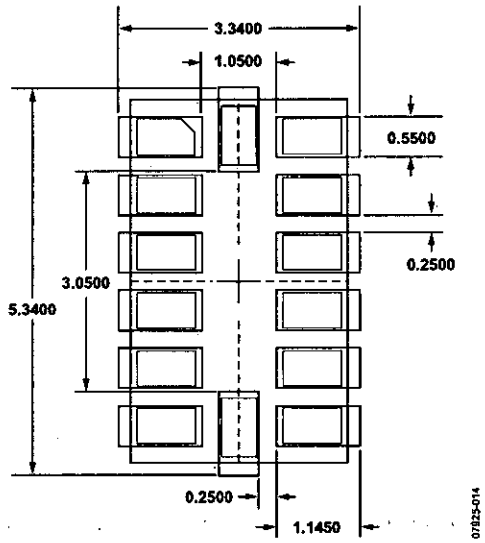


Figure 18. Recommended Printed Wiring Board Land Pattern (Dimensions shown in millimeters)

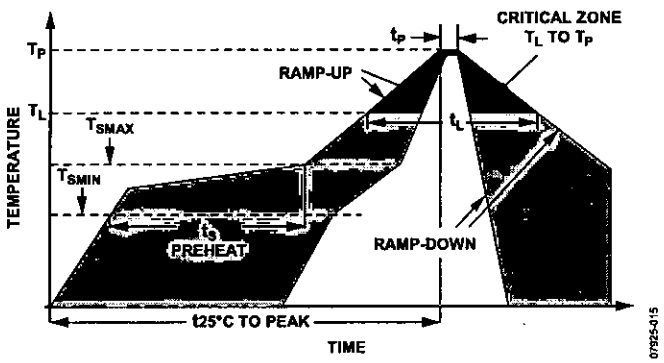


Figure 19. Recommended Soldering Profile

Table 21. Recommended Soldering Profile[1, 2]

	Condition	
Profile Feature	**Sn63/Pb37**	**Pb-Free**
Average Ramp Rate from Liquid Temperature (T_L) to Peak Temperature (T_P)	3°C/sec max	3°C/sec max
Preheat		
Minimum Temperature (T_{SMIN})	100°C	150°C
Maximum Temperature (T_{SMAX})	150°C	200°C
Time from T_{SMIN} to T_{SMAX} (t_S)	60 sec to 120 sec	60 sec to 180 sec
T_{SMAX} to T_L Ramp-Up Rate	3°C/sec max	3°C/sec max
Liquid Temperature (T_L)	183°C	217°C
Time Maintained Above T_L (t_L)	60 sec to 150 sec	60 sec to 150 sec
Peak Temperature (T_P)	240 + 0/−5°C	260 + 0/−5°C
Time of Actual T_P − 5°C (t_P)	10 sec to 30 sec	20 sec to 40 sec
Ramp-Down Rate	6°C/sec max	6°C/sec max
Time 25°C to Peak Temperature	6 minutes max	8 minutes max

[1] Based on JEDEC Standard J-STD-020D.1.
[2] For best results, the soldering profile should be in accordance with the recommendations of the manufacturer of the solder paste used.

.INE DIMENSIONS

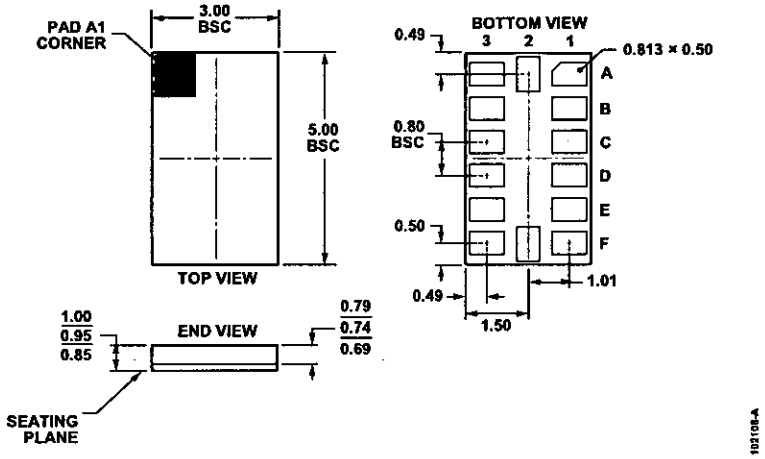


Figure 20. 14-Terminal Land Grid Array [LGA]
(CC-14-1)
Solder Terminations Finish Is Au over Ni
(Dimensions shown in millimeters)

.RING GUIDE

	Measurement Range (*g*)	Specified Voltage (V)	Temperature Range	Package Description	Package Option
45BCCZ[1]	±2, ±4, ±8, ±16	2.5	−40°C to +85°C	14-Terminal Land Grid Array [LGA]	CC-14-1
45BCCZ-RL[1]	±2, ±4, ±8, ±16	2.5	−40°C to +85°C	14-Terminal Land Grid Array [LGA]	CC-14-1
45BCCZ-RL7[1]	±2, ±4, ±8, ±16	2.5	−40°C to +85°C	14- Terminal Land Grid Array [LGA]	CC-14-1
DXL345Z[1]				Evaluation Board	
DXL345Z-M[1]				Analog Devices Inertial Sensor Evaluation System, Includes ADXL345 Satellite	
DXL345Z-S[1]				ADXL345 Satellite, Standalone	

IS Compliant Part.

AT commands

Factory default setting:

Name: HMSoft; Baud: 9600, N, 8, 1; Pin code: 000000; Peripheral Role; transmit mode.

AT Command format:

Uppercase AT command format. string format, without any other symbol. (e.g. \r or \n).

On Transmit version: Only accept AT Command from UART interface when Bluetooth device is not connected with remote device.

On Remote version: Can accept AT Command from UART interface when Bluetooth Device is not connected with remote device, Also can accept AT Command from remote Bluetooth device when connected that.

On PIO collection version: Only accept AT Command from UART interface when Bluetooth device is not connected with remote device.

1. Test Command

Send	Receive	Parameter
AT	OK OK+LOST	None

If Module is not connected to remote device will receive: "OK".

If Module has connected, module will disconnected from remote device, if "AT + NOTI" is setup to 1, will receive: "OK+LOST".

2. Query ADC conversion value

Send	Receive	Parameter
AT+ADC[para]	OK+Get:0.00	para: 3~B Map to PIO3~PIO8

3. Query module MAC address

Send	Receive	Parameter
AT+ADDR?	OK+ADDR:MAC Address	None

4. Query/Set Advertising type

Send	Receive	Parameter
AT+ADTY?	OK+Get:[para]	None
AT+ADTY[para]	OK+Set:[para]	para:0~3 0:Connect by any device 1:Allow to connect with last succeeded device(within 1.28s after power on) 2:Allow to broadcast and scanning 3:Only advertising Default:0

Added since V519 version

5. Query/Set ANCS switch

Send	Receive	Parameter
AT+ANCS?	OK+Get:[para]	None
AT+ANCS[para]	OK+Set:[para]	para: 0, 1 0: Off 1: On

NOTE: 1. Must execute AT+TYPE3 first

2. Please send AT+RESET to restart module if you set value 1

3. Added in V524 version

6. Query/Set whitelist switch(only allow 3 mac address link to module)

Send	Receive	Parameter
AT+ALLO?	OK+Get:[para]	None
AT+ALLO[para]	OK+Set:[para]	para: 0, 1 0: Off 1: On Default: 0

Added in V523 version

7. Query/Set whitelist MAC address

Send	Receive	Parameter
AT+AD[para1]??	OK+AD[para1]?:[para2]	None
AT+AD[para1][para2]	OK+AD[para1][para2]	para1: 1, 2, 3 para2: MAC address

e.g.

Query whitelist MAC address 1

Send: AT+AD1??

Receive: OK+AD1001122334455 (001122334455 is MAC address)

Set whitelist MAC address 2

Send: AT+AD2001122334455 (001122334455 is MAC address)

Receive: OK+AD2001122334455

8. Query/Set Advertising interval

Send	Receive	Parameter
AT+ADVI?	OK+ Get:[para]	None
AT+ADVI[para]	OK+ Set:[para]	para: 0 ~ F 0: 100ms 1: 152.5ms 2: 211.25ms

		3: 318.75ms 4: 417.5ms 5: 546.25ms 6: 760ms 7: 852.5ms 8: 1022.5ms 9: 1285ms A: 2000ms B: 3000ms C: 4000ms D: 5000ms E: 6000ms F: 7000ms Default: 0 HMSoft Default: 0 HMSensor Default: 9

The maximum 1285ms recommendations form the IOS system. That is to say, 1285ms is apple allowed, but in response to scan and connected all the time will be long.

Added since V515 version. V521 allows max value to be 9, V522 allows max value to be F.

9. Query/Set module PIO output state, after power supplied

Send	Receive	Parameter
AT+BEFC?	OK+ Get:[para]	None
AT+BEFC[para]	OK+ Set:[para]	para: 000 ~ 3FF Default: 000

14. Query/Set baud rate

Send	Receive	Parameter
AT+BAUD?	OK+Get:[para]	para: 0~8
AT+BAUD[para]	OK+Set:[para]	0: 9600 1: 19200 2: 38400 3: 57600 4: 115200 5: 4800 6: 2400 7: 1200 8: 230400 Default: 0(9600)

Need to re-power module.

e.g.

Query baud rate:

Send: AT+BAUD?

Receive: OK+Get:0

Set baud rate to 19200:

Send: AT+BAUD1

Receive: OK+Set:1

Note: If set baud rate to 7(1200), after next power on, module will not support any AT commands, until PIO0 is pressed, then module will change baud rate to 9600.

20. Try to connect to last succeeded device

Send	Receive	Parameter
AT+CONNL	OK+CONN[para]	para: L, E, F, N L: Connecting E: Connect error F: Connect Fail N: No Address

NOTE: Only central role is used. Module must setup AT+ROLE1, AT+IMME1 first.

If remote device has already connected to other device or shut down, "OK+CONNF" will received after about 10 seconds.

21. Try to connect to device with given address

Send	Receive	Parameter
AT+CONN[para1]	OK+CONN[para2]	para1: 0~5 para2: A , E, F A: Connecting E: Connect error F: Connect Fail

Notice: Only central role is used. Module must setup AT+ROLE1, AT+IMME1, AT+DISC? first.

If remote device has already connected to other device or shut down, "OK+CONNF" will received after about 10 Seconds.

24. Clear last connected device address

Send	Receive	Parameter
AT+CLEAR	OK+CLEAR	None

36. Query/Set Module work type

Send	Receive	Parameter
AT+IMME?	OK+Get:[para]	para: 0, 1 1: When module is powered on, only respond the AT Command, don' t do anything until AT + START is received,or can use AT+CON,AT+CONNL 0: When power on, work immediately Default: 0
AT+IMME[para]	OK+Set:[para]	

NOTE: Only used for central role. Need to re-power module.

45. Query/Set Module Work Mode

Send	Receive	Parameter
AT+MODE?	OK+Get:[para]	para: 0, 1, 2 0: Transmission Mode 1: Transmission Mode + PIO Collection Mode 2: Transmission Mode + Remote Control Default: 0
AT+MODE[para]	OK+Set:[para]	

Mode 0:
Before establishing a connection, you can use the AT command configuration module through UART.
After established a connection, you can send data to remote side from each other.

Mode 1:
Before establishing a connection, you can use the AT command configuration module through UART.
After established a connection, you can send data to remote side. Remote side can do fellows:

A. Send AT command configuration module.
B. Collect PIO04 to the PIO11 pins input state of HM-10.
C. Collect PIO03 pins input state of HM-11.
D. Remote control PIO2, PIO3 pins output state of HM-10.
E. Remote control PIO2 pin output state of HM-11.
F. Send data to module UART port (not include any AT command and per package must less than 20 bytes).

Mode 2:
Before establishing a connection, you can use the AT command configuration module through UART.
After established a connection, you can send data to remote side. Remote side can do fellows:
A. Send AT command configuration module.
B. Remote control PIO2 to PIO11 pins output state of HM-10.
C. Remote control PIO2, PIO3 pins output state of HM-11.
D. Send data to module UART port (not include any AT command and per package must less than 20 bytes).

48. Query/Set Module name

Send	Receive	Parameter
AT+NAME?	OK+NAME[para]	para: module name, Max length is 11. Default: HMSoft
AT+NAME[para]	OK+Set[para]	

e.g.

Change module name to bill_gates

Send: AT+NAMEbill_gates

Receive: OK+Set:bill_gates

49. Query/Set Parity bit

Send	Receive	Parameter
Query: AT+PARI?	OK+Get:[para]	None
Set: AT+PARI[para]	OK+Set:[para]	para: 0,1,2 0:None 1:EVEN 2:ODD Default: 0

Need to re-power module.

55. Query/Set Module Power

Send	Receive	Parameter
AT+POWE?	OK+Get:[para]	None
AT+ POWE [para]	OK+Set:[para]	para: 0 ~ 3 0: -23dbm 1: -6dbm 2: 0dbm 3: 6dbm Default: 2

59. Restart module

Send	Receive	Parameter
AT+RESET	OK+RESET	None

60. Query/Set Master and Slaver Role

Send	Receive	Parameter
AT+ROLE?	OK+Get:[para]	para: 0, 1 0: Peripheral 1: Central Default: 0
AT+ROLE[para]	OK+Set:[para]	

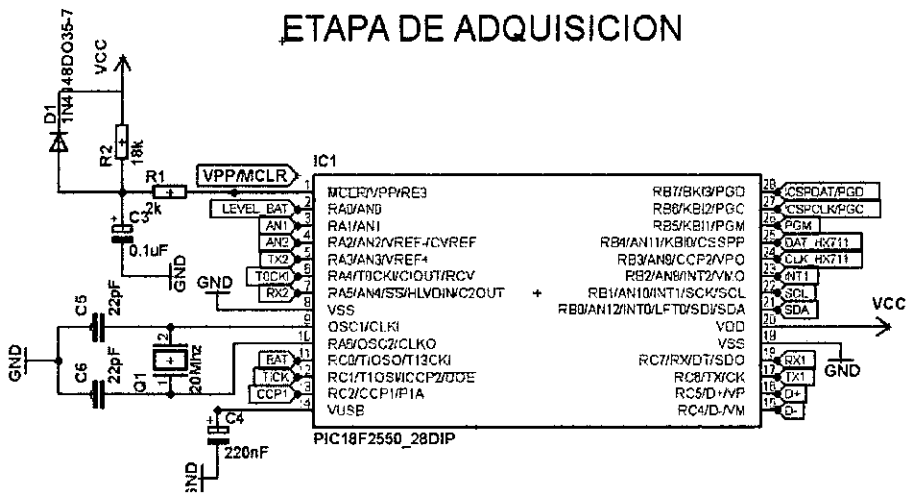
CÓDIGO ESTADOUNIDENSE ESTÁNDAR PARA EL INTERCAMBIO DE INFORMACIÓN

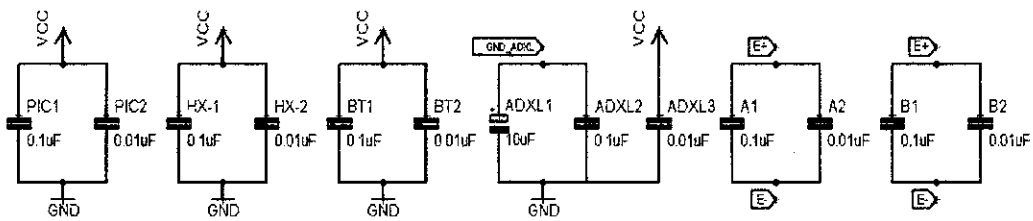
Caracteres de control ASCII			
DEC	HEX	Símbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de línea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

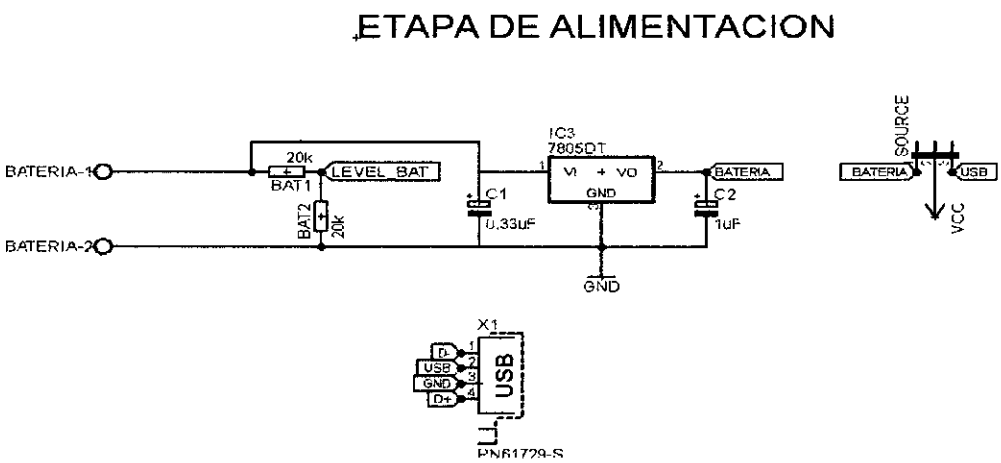
Caracteres ASCII imprimibles											
DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
32	20h	espacio	64	40h	@	96	60h	`			
33	21h	!	65	41h	A	97	61h	a			
34	22h	"	66	42h	B	98	62h	b			
35	23h	#	67	43h	C	99	63h	c			
36	24h	$	68	44h	D	100	64h	d			
37	25h	%	69	45h	E	101	65h	e			
38	26h	&	70	46h	F	102	66h	f			
39	27h	'	71	47h	G	103	67h	g			
40	28h	(	72	48h	H	104	68h	h			
41	29h	)	73	49h	I	105	69h	i			
42	2Ah	*	74	4Ah	J	106	6Ah	j			
43	2Bh	+	75	4Bh	K	107	6Bh	k			
44	2Ch	,	76	4Ch	L	108	6Ch	l			
45	2Dh	-	77	4Dh	M	109	6Dh	m			
46	2Eh	.	78	4Eh	N	110	6Eh	n			
47	2Fh	/	79	4Fh	O	111	6Fh	o			
48	30h	0	80	50h	P	112	70h	p			
49	31h	1	81	51h	Q	113	71h	q			
50	32h	2	82	52h	R	114	72h	r			
51	33h	3	83	53h	S	115	73h	s			
52	34h	4	84	54h	T	116	74h	t			
53	35h	5	85	55h	U	117	75h	u			
54	36h	6	86	56h	V	118	76h	v			
55	37h	7	87	57h	W	119	77h	w			
56	38h	8	88	58h	X	120	78h	x			
57	39h	9	89	59h	Y	121	79h	y			
58	3Ah	:	90	5Ah	Z	122	7Ah	z			
59	3Bh	;	91	5Bh	[	123	7Bh	{			
60	3Ch	<	92	5Ch	\	124	7Ch	|			
61	3Dh	=	93	5Dh	]	125	7Dh	}			
62	3Eh	>	94	5Eh	^	126	7Eh	~			
63	3Fh	?	95	5Fh	_	eCodigoASCII.com.ar					

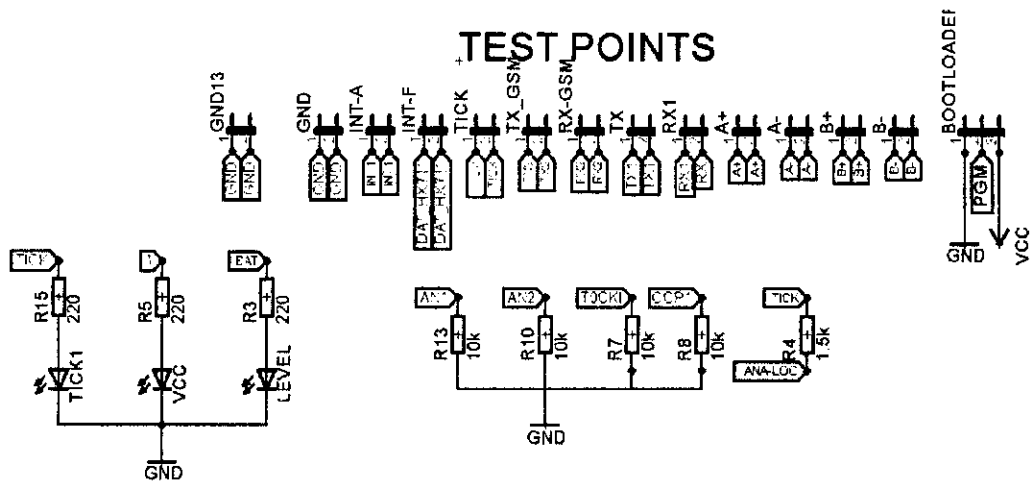
ASCII extendido											
DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	▒	208	D0h	ð	240	F0h	
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ö	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	▌	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

DISEÑO DE PLACA DE COMUNICACIONES SLAVE EN EAGLE

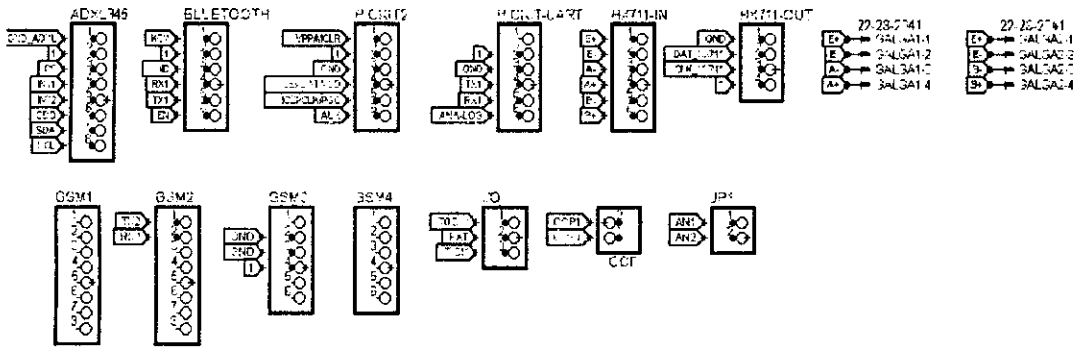


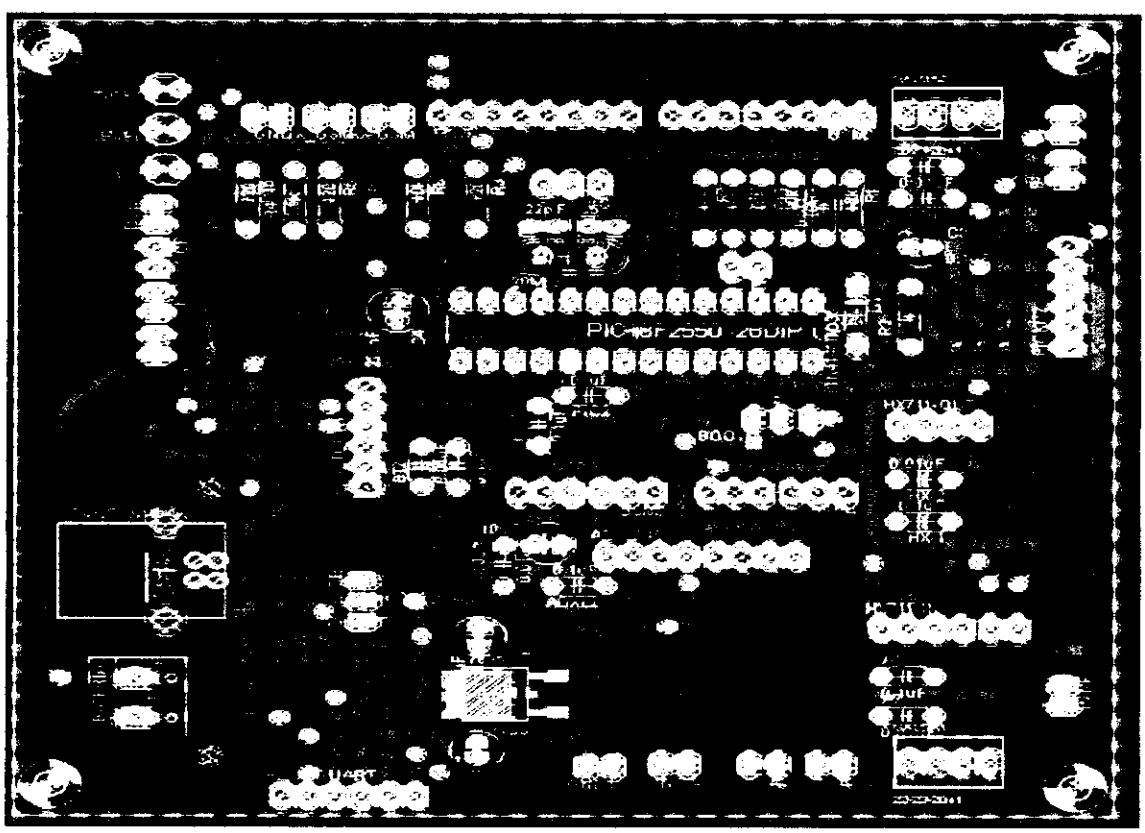






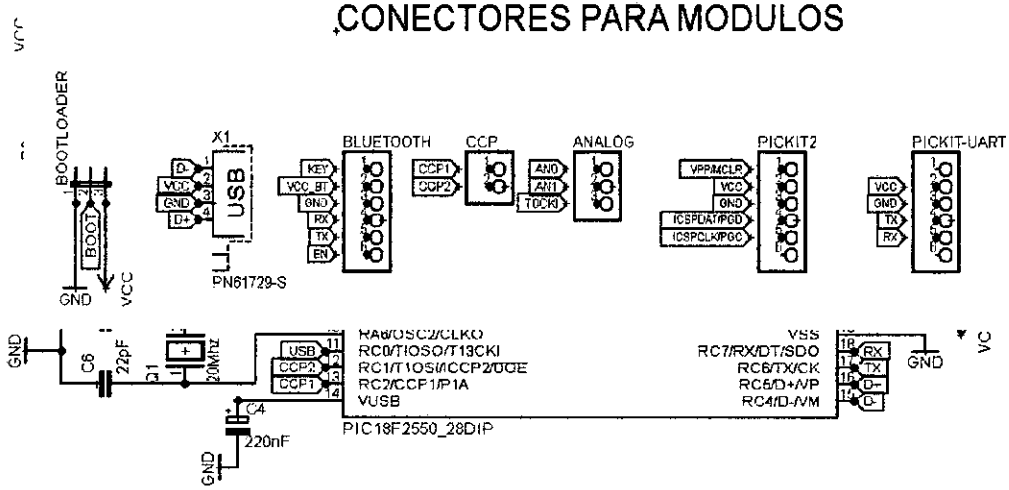
CONECTORES PARA MODULOS



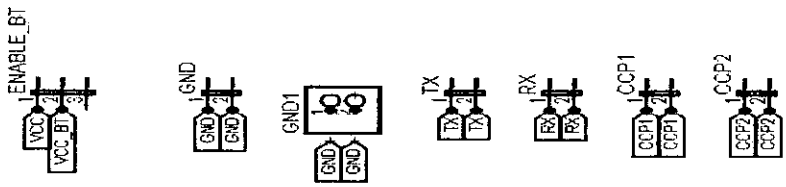


DISEÑO DE PLACA DE COMUNICACIONES GATEWAY EN EAGLE

CONECTORES PARA MODULOS



TEST POINTS



INDICADORES